Dr. Dobb's Journal

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS #105 JULY 1985 $2.95 (3.95 CANADA)

Hacking Hardware

Build the Ultimate Print Spooler

Add a Real-Time Clock

Make MSDOS Work Like Unix

Turbo Pascal vs. the Standard

Software development isn't a mountainous task once you eliminate the high C errors.

When you can find and fix bugs at the earliest possible moment, creating software stops being such an uphill grind.

And the Smart/C Environment makes it possible. It's a complete, fully-integrated development environment for C that saves you from the creativity-inhibiting cycle of edit, compile, re-edit, re-compile, link, load, test, re-edit, re-compile, etc., ad infinitum. Smart/C puts the fun back in programming, because you spend your time creating... not waiting.

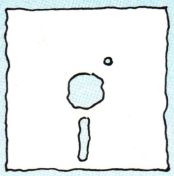
Here's why. Syntax errors are eliminated automatically as code is entered. Smart/C's highly integrated editor and interpreter allow you to interpret your program at any time in the creation process, so logic errors can be ferreted out as soon as the algorithm exists—long before any compile, link, or load.

The complete integration of the editor and interpreter means you can stop anywhere in the interpret cycle, edit, and then go right back into the interpreter exactly where you left off. Not only that, the screen-oriented user interface lets you see all operations, even interpretation, right on the listing of the code.

Circle **no. 4** on reader service card.

And to make maintenance programming easier, Smart/C's Migrator allows existing C code produced with any editor to be modified and run within the Smart/C Environment.

All of which makes Smart/C an excellent tool. It's flexible, non-restrictive, and lets you create elegant, readable, error-free programs that you can watch run with a great feeling of satisfaction.



Smart/C™

Free Demo Disk!

To fully appreciate Smart/C, you have to see it in action. For your free IBM PC MS-DOS demo disk, call us. Or write us on your company letterhead.
AGS Computers, Inc., Advanced Products Division, 1139 Spruce Drive, Mountainside, NJ 07092.
800-AGS-1313. In NJ, 201-654-4321.

Smart/C Features

The Smart/C Environment
- ☐ Fully integrated editor and interpreter
- ☐ Only one load brings them both in
- ☐ One command set
- ☐ Move between one another at will

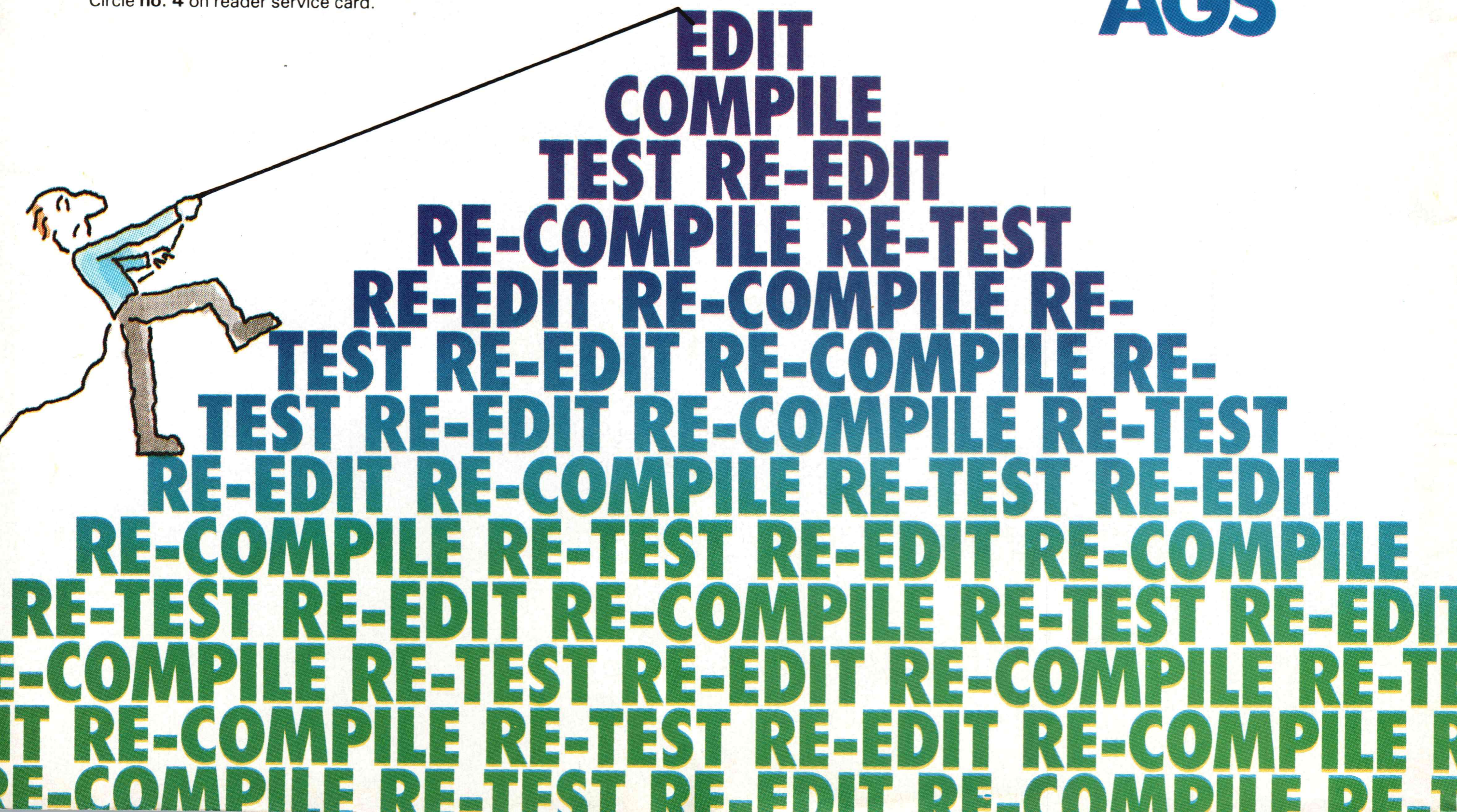
Syntax Directed Editor
- ☐ vi-like command set
- ☐ Automatically provides formats for blocks, *for, case* and *if* statements

Interpreter
- ☐ Current module can call external modules during interpretation
- ☐ Has Include capability
- ☐ Totally precompilation—no incremental compile
- ☐ Can interpret partially defined files allowing for rapid prototyping
- ☐ Variable speed of interpretation
- ☐ Multiple windows with user-defined sizes

The Smart/C Migrator
- ☐ Allows C code produced with any editor to be interpreted by Smart/C
- ☐ Reformats for readability

Smart/C has been ported to UNIX™ System V Release 2, Berkeley 4.2, Xenix,™ and MS-DOS. Versions run on 8086- and 68000-based machines, as well as proprietary architectures. Smart/C runs on PCs, micros, supermicros, minis, and even mainframes.

AGS



Introducing the MIX Editor

(with Split Screen - both horizontal and vertical)

A Powerful Addition To Any Programmer's Tool Box

- Full Screen Editing
- WordStar Key Layout
- Custom Key Layouts
- Terminal Configuration
- Help Files
- Backup Files

Introductory Offer Only

29.95

30 Day Money Back Guarantee

- Programmable
- Macro Commands
- Custom Setup Files
- Mnemonic Command Mode
- Multiple File Editing
- Split Screen Editing

For PCDOS/MSDOS (2.0 and above/128K) • IBM PC/Compatibles, PC Jr., Tandy 1000/1200/2000, & others
For CPM80 2.2/3.0 (Z80 required/64K) • 8" SSSD, Kaypro 2/4, Osborne I SD/DD, Apple II, & others

Great For All Languages

A general purpose text processor, the MIX Editor is packed with features that make it useful with any language. It has auto indent for structured languages like Pascal or C. It has automatic line numbering for BASIC (255 character lines). It even has fill and justify for English.

Terminal Configuration

A utility for defining terminal features (smart features included) allows the editor to work with any terminal. Over 30 of the most popular terminals are built-in.

Custom Key Layouts

Commands are mapped to keys just like WordStar. If you don't like the WordStar layout, simply change it. Any key can be mapped to any command. You can also define a key to generate a string of characters, great for entering keywords.

Split Screen

You can split the screen horizontally or vertically and edit two files simultaneously.

Macro Commands

The MIX Editor allows a sequence of commands to be executed with a single keystroke. You can define a complete editing operation and perform it at the touch of a key.

MIX software
2116 E. Arapaho
Suite 363
Richardson, Tx 75081
(214) 783-6001

MSDOS is a trademark of Microsoft
PCDOS is a trademark of IBM
CPM80 is a trademark of Digital Research
WordStar is a trademark of MicroPro

Custom Setup Files

Custom keyboard layouts and macro commands can be saved in setup files. You can create a different setup file for each language you use. The editor automatically configures itself using a setup file.

Command Mode

Command mode allows any editor command to be executed by name. It is much easier to remember a command name versus a complicated key sequence. Command mode makes it easy to master the full capability of the editor. Frequently used commands can be mapped to keys. Infrequent commands can be executed by name.

Editor Commands

The editor contains more than 100 commands. With so many commands, you might think it would be difficult to use. Not so, it is actually extremely simple to use. With command mode, the power is there if you need it, but it doesn't get in your way if you don't. Following is a list of some of the commands.

Cursor Commands

Left/Right/Up/Down
Tab Right/Tab Left
Forward Word/Backward Word
Beginning of Line/End of Line
Scroll Up/Scroll Down
Window Up/Window Down
Scroll Left/Scroll Right
Top of File/Bottom of File
• • •

Block Commands

Copy/Move/Delete
Read/Write
Lower Case/Upper Case
Fill/Justify
Print

File Commands

Directory (with wild cards)
Show File/Help File
Input/Output File
Delete File/Save File

Other Commands

Split Screen/Other Window
Find String/Replace String
Replace Global/Query Replace
Delete Line/Undelete Line
Delete Word/Undelete Word
Insert Mode/Overwrite Mode
Open Line/Join Line
Duplicate Line/Center Line
Set Tab/Clear Tab
• • •

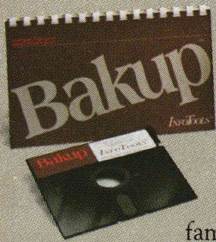
To Order: **Call Toll Free 1-800-523-9520,** (Texas only **1-800-622-4070)**
Mix Editor ____ $29.95 + shipping ($5 USA/$10 Foreign) Texas residents add 6% sales tax

Visa ____ MasterCard ____ Card # ____________ Exp. Date ________
COD ____ Check ____ Money Order ____ Disk Format ____________
Computer ____________ Operating System: MSDOS ____ PCDOS ____ CPM80 ____
Name ____________
Street ____________
City/State/Zip ____________
Country ____________
Phone ____________

MIX software
2116 E. Arapaho
Suite 363
Richardson, Tx 75081
Dealer Inquiries Welcome
Call (214) 783-6001

DD

TeamMate's Storage Solution: Two Minute Backup and Versatility



Two minutes to back up data?

It's true with TeamMate's family of storage subsystems featuring the remarkable Kodak™ 2.78 megabyte flexible drive and **free Bakup**™ software.

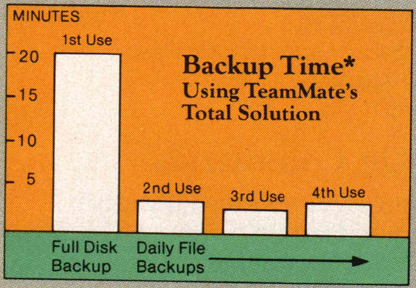
Frankly, most people know they need to protect their valuable data by backing up files regularly. They don't because it takes too long.

TeamMate has taken care of that.

All you do is back up your full hard disk once. 10 megabytes can be copied in about 20 minutes onto 3 or 4 Kodak high capacity diskettes.

Then TeamMate's easy-to-use Bakup software allows you to automatically back up your modified files in as little as 2 minutes!

New advances in backup are just part of the TeamMate/Kodak success story.



TeamMate's Winchester and Kodak 2.78MB flexible drive subsystems also add unprecedented flexibility and versatility to the IBM™ PC, AT, or compatible computers.

Kodak Drives... Unlimited Storage at a Low Cost

Install them internally or externally. Either way, TeamMate's family of Kodak 2.78MB flexible drives give you Winchester-like capacity, plus the convenient familiarity of a floppy. You get eight times the online storage capacity of a standard floppy... and unlimited expandability.

The Kodak 2.78MB drive guarantees media interchangeability from drive to drive, and it can also read standard IBM PC and AT diskettes—ideal for transporting your AT's data to standard PC's.

Need to achieve high capacity storage and rugged portability? Use our TeamMate 1103 (single Kodak drive) or the 1104, a Kodak drive and standard floppy combination—perfect for upgrading the XT. Or, use our external models featuring one or two Kodak drives to backup several PC's.

Winchester/Kodak Combinations...More Versatility

TeamMate's combined Winchester and Kodak subsystems deliver the greatest versatility available today. Use the high speed Winchester, and back up your data with the Kodak flexible drive. Also use the Kodak for data security, diskette savings, or data archiving.

Installation takes only minutes...and there's no need for software drivers or patches.

The TeamMate 1123 installs internally and adds 20 megabytes of hard disk storage plus the Kodak flexible drive. External Winchester/Kodak combinations turn a minimum PC into a sophisticated supersystem.

Save...With TeamMate Winchester Subsystems

High performance and a variety of capacities. That's TeamMate's family of Winchester subsystems.

Add 10 or 20 megabytes of storage to your system either internally or externally. For really big jobs, look to TeamMate's 1132...32 megabytes of formatted add-in storage.

Enhance a basic IBM PC AT configuration with 20 or 32 megabytes of reliable Winchester storage. TeamMate 1120 AT & 1132 AT attach directly to the AT's existing controller. Complete with drive, cable, mounting slides, and Bakup software, they're ready to go in just minutes.

Adding a TeamMate to your system is the cost-effective way to handle a variety of tasks.

Call TeamMate today for more information.

Be sure to ask us about our Apple family.

Only TeamMate's Kodak Drive Solves All Your System Needs	
✓ Hard Disk Backup	High speed full or selective file backup onto high capacity diskettes. 10MB hard disk needs only 4 Kodak diskettes.
✓ Primary Storage	2.78 megabytes of online random access storage
✓ Portability	Floppy ruggedness...ideal for portable computers
✓ Expandability	Unlimited storage capacity—just change diskettes
✓ Data Security	Removeability gives you ultimate data security without security software
✓ Interchangeability	Media interchangeability guaranteed from drive to drive
✓ Read Standard Diskettes	Reads standard diskettes so you can exchange data between IBM PCs and ATs
✓ Diskette Savings	One Kodak 2.78MB diskette replaces 8 standard IBM floppies.
✓ Convenience	Install in minutes using standard DOS software

Team·Mate

TeamMate by Data Technology
2525 Walsh Avenue
Santa Clara, CA 95051
Tel: (408) 986-9545
Telex: 4940763 DTCSC

™ Kodak is a registered trademark of Eastman Kodak Corporation.
™ IBM is a registered trademark of International Business Machines Corporation.
™ Bakup is a registered trademark of InfoTools Corporation.

Circle **no. 3** on reader service card.

Dr. Dobb's Journal (USPS 307690) is published monthly by M&T Publishing, Inc., 2464 Embarcadero Way, Palo Alto, CA 94303, (415) 424-0600. Second class postage paid at Palo Alto and at additional entry points.

Address correction requested. Postmaster: Send Form 3579 to *Dr. Dobb's Journal*, P.O. Box 27809, San Diego, CA 92128. **ISSN 0278-6508**

Subscription Rates: $25 per year within the United States, $46 for airmail to Canada, $62 for airmail to other countries. Foreign subscriptions must be pre-paid in U.S. Dollars, drawn on a U.S. Bank. For subscription problems, call: outside of CA 800-321-3333; within CA 619-485-9623 or 566-6947.

Foreign Distributor: Worldwide Media Service, Inc., 386 Park Ave. South, New York, NY 10016, (212) 686-1520 TELEX: 620430 (WUI)

People's Computer Company

Dr. Dobb's Journal is published by M&T Publishing, Inc. under license from People's Computer Company, 2682 Bishop Dr., Suite 107, San Ramon, CA 94583, a non-profit, educational corporation.

July 1985
Volume 10, Issue 7

CONTENTS

Referees

Michael Enright
Robert Tripp

Cover

The cover cartoon by Rand Renfroe was colored on a Via Video Graphics System by Doyle Puppo and Associates.

Editor's Note

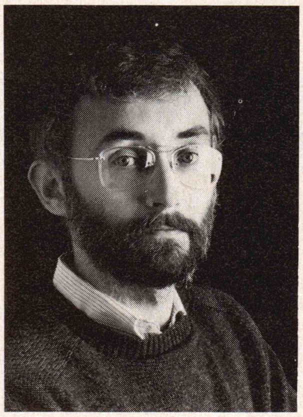
In C chest this month Allen Holub offers an MSDOS version of the Unix directory utility ls. As the magazine went into production, we discovered that the program did not operate exactly as promised. The command ls /, which was supposed to list the contents of the root directory from another directory, did not function properly under DOS 2.x. Also, when run on a Compaq, ls did not underline directories, but printed them in half intensity. This is probably due to the particular ANSI.SYS file used.—Ed.

Dr. Dobb's Journal

ARTICLES

COLUMNS

DEPARTMENTS



We do this for you.

"We:" that's us, the editorial staff; "this" is the magazine. It's that "you" that makes things tricky.

"You" are a statistical construct. You are 40,000 readers with at least 40,000 opinions on any random subject, 40,000 ideas about what *Dr. Dobb's Journal* should be.

You are a systems programmer for IBM. A DoD Ada specialist. You are the DP Manager for a large corporation. You are the only programmer for a small firm. You are an independent software developer. A consultant. You're a structural engineer who happens to write software. A researcher. A teacher. A student.

You program in a high-level language because you're a high-level thinker, and you use assembly language only to tighten the bolts. You have concluded that all serious computer languages are fundamentally equivalent, so why doesn't everyone just use Pascal? You think that computer languages differ widely in power, efficiency and naturalness, so why doesn't everyone use Forth (since, as surely everyone knows, Forth code is invariably faster and tighter and more readable than anything else)? You've forgotten in the past year that there is any high-level language but C.

You refuse to program in a high-level language, pouring all your creative efforts into high-torque 68K assembly language code. 8086/8088 code. Z80/8080 code. You know the 6502 inside out. You know, even if you are in the vanishing minority, that the 6809 is all the processor anyone really needs. You know things about the 80286 that they haven't discovered at Intel yet.

You know what you like about *DDJ*. The *DDJ* articles for 1985 that you were most interested in (so far) included pieces dealing with the Unix, CP/M and MSDOS operating systems; the C and Prolog languages; the IBM PC, the PC/AT, the Mac, the Commodore 64, Z80 machines and "machines in the 68000/16000 class with virtual memory."

You're intelligent, knowledgeable, creative and opinionated, but let's face it: you are not particularly consistent. Neither, consequently, are we.

This is the hardware issue of *Dr. Dobb's Journal* of "software tools for advanced programmers." Why are we publishing hardware construction articles in a software magazine?

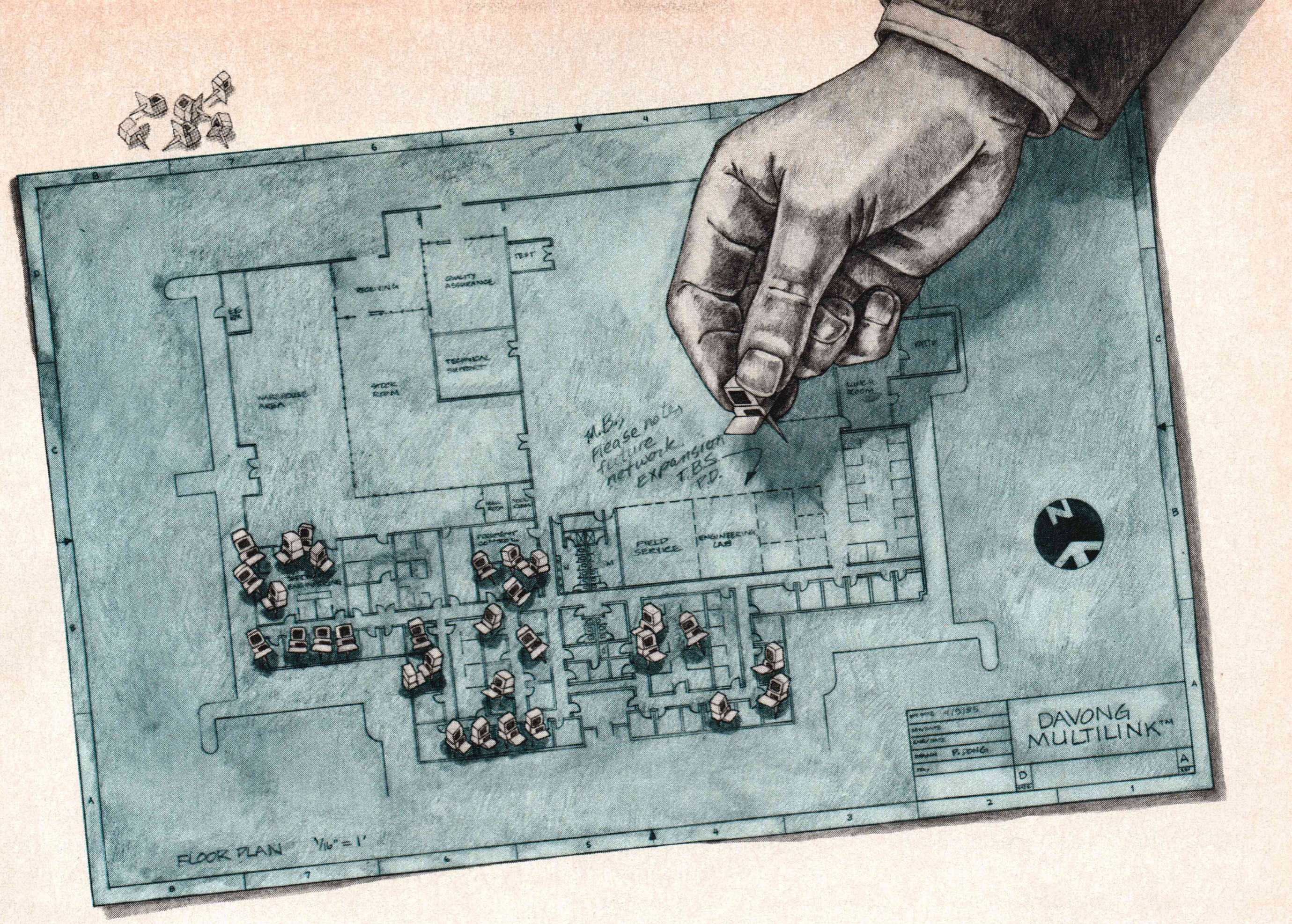
We are doing it, of course, for you. The "Fatten Your Mac" article in January was one of our most popular articles to date, and other hardware-related pieces have been well-received.

We have a broad constituency, trying as we do to address all advanced programmers, and magazine space is heartbreakingly limited. So we try to be as broad as possible without sacrificing depth. We publish generalizable code whenever we can, emphasizing good algorithms.

But the canvas tears when stretched to include hardware articles. Although there is much non-hardware material in this issue, we have this month committed a sizeable portion of the magazine to hardware-related material. We offer this collection of hardware articles with a question: how would you like to see us cover hardware in the future? With an occasional article when it seems appropriate? With an annual hardware issue like this? Or a hardware issue of a different sort? In separate monographs/project papers, leaving the magazine pages for software only? Or not at all?

Michael Swaine

Michael Swaine



Pinpointing Your Network Solution

Let's get straight to the point! If you want a powerful network to link up to 255 IBM® and IBM-compatible computers, then Davong MultiLink™ is clearly the engineered choice for you.

Let's pin down a few facts.

You don't have to be a networking wizard with MultiLink. Menus and help screens make it easy for you to control the network—not vice versa. Managing the space on one or 255 hard disks is simple.

You have worry-free expand-ability with Davong MultiLink. Nobody has to stop work just because you want to add another workstation or file server to your MultiLink network—even if you add a Davong DataSystem™ to make a file server of your PC. And, file servers also perform double-duty as workstations.

You don't have to worry about security. MultiLink provides password protection and three levels of access for each volume. There's even password protection for each hard disk! In addition, you can share volumes without collision worries. The shared DOS feature takes care of that. And, automatic file-locking capability and optional record locking. Finally! A network that provides true file-sharing!

You can have a unique boot configuration—directly from the network. No need for boot diskettes. Use space from any hard disk on the network. Keep all your files in one volume or spread them among several. MultiLink allows access to 6 active volumes at a time. And, accessing other hard disk volumes is easy.

Included with Davong MultiLink is one of the most complete print spooling packages on the market today. Print to any serial or parallel printer on the network—even multiple copies. It's easy with MultiLink.

Interested? Find out more about Davong MultiLink by contacting your local Davong dealer.

Why be stuck with less?

DAVONG

Davong Systems, Inc.
217 Humboldt Court
Sunnyvale, CA. 94089
Phone: (408) 734-4900
Telex: 176386

Circle **no. 2** on reader service card.

Small C

Dear *DDJ*:

I don't know if anybody's still keeping track of these things, but I've found a Small C bug. The source

```
int 1ptr; /* global integer */
...
routine( ) {
    int r;
    char bytes[2];
    ...
    1ptr += r - bytes;
    ...
```

compiled to (Z80 mnemonics)

```
... (usual stuff)
LD HL, (LPTR)
PUSH HL;;;    (save left side of
                += statement)
LD HL,4
ADD LH,SP
CALL CCGINT
EX DE,HL;;
LD HL,2
ADD HL,SP
CALL CCSUB
POP DE        (retrieve left side of
                += statement)
ADD HL,DE     (add it)
LD A,L        (forget what's going
                on ....)
LD (DE),A     (should be LD
                (LPTR),HL)
...
```

which is entirely the wrong idea and has very nasty effects (the bug also occurs with local destinations for the assignment, but with somewhat less dangerous results). The problem disappeared when I *removed* the last code line from plung2( ) (aka plnge2 in "The Small-C Handbook" by James E. Hendrix) in cc31.c:

```
if ((oper==ffsub) |
    (oper==ffadd))
    result(lval, lval2);
```

In my copy of the compiler, this is the only place result( ) is used, so I removed it from cc33.c. The compiler compiled itself in this shape, and successfully compiled a reasonably large program, so maybe this is the correct fix, but I'm not deeply convinced.

Fixing the compiler is a real adventure for me, because I don't know how it works. I do wonder what solemn oath bound the original authors, so that no trace of comment is allowed to stain the pure pages of code . . . .

Cordially,
Gregor Owen
μ Software/Hardware
35 Admiral St.
Port Jefferson Station,
New York, NY 11776

Other fixes and a status update for Small C will appear in the August issue—Ed.

Iconoclasm

Dear *DDJ*:

By your response to Frank Gaude's criticism of icons, it appears as though you don't fully appreciate his point. In fact, your apparent praise of 'active, growing icons' only strengthens his argument. What will we have if software developers everywhere start creating systems with user-definable icons? We will have the second stage in the evolution of language, as icons were the first. What is more, we will have multitudinous languages, with each programmer or user designing the icon interface to [his] own personal taste. Nobody will be able to sit down at somebody else's system and run it. We will have the Tower of Babel all over again. Confusion. Insanity. Let's not do it all over again.

For those who insist upon performing all operations with a single keystroke, there are menus.

Another thing: metaphors. This isn't really related to icons, except that it's tangled up in your discussion of them. I wouldn't quibble with anyone who said that the purpose of software was to entertain, educate, or to simply get the job done . . . or to obfuscate, mislead, and lie. But 'The purpose of software is to realize metaphors'? Could you say that again, please? Could you illustrate with an example?

Well, gee, now that I've gotten going it's just too hard to stop. One more thing. I greatly doubt that all of your readers know that Richard Conn wrote ZCPR3. As a consequence, I would have thought it to be a prudent editorial decision to add a note to that effect to his review [*DDJ* #103, May 1985] of the Ampro computers, which comes bundled with ZCPR3. What do you think?

(By the by, I mean no criticism of ZCPR3. It's the best damn thing to come down the pike since the Z80. Conn—and Echelon—are to be applauded roundly.)

Sincerely,
Dreas Nielsen
234 NW 30th St.
Corvallis, OR 97330

Indeed we should have identified Richard Conn's affiliation.—Ed.

CP/M

Dear *DDJ*:

I have been using CP/M 2.2 for several years, and acquired CP/M Plus about 9 months ago. I am pleased with it, but I have run up against one difficulty, namely that the SAVE program does not correctly save .REL files (produced by Microsoft's BASCOM) loaded by MICROSOFT's L80 relocating loader. I wrote a short BA-

SIC program to list ASCII files, including line numbers; after finishing a listing it prompts the operator for another file name. L80 loads the .REL file, and shows a starting address of 013CH. (The starting address of CP/M programs is 0100H.) If 013C is used as the starting address when the program is saved (using CP/M Plus's SAVE program) it does not run properly. It returns to the operating system after a few seconds, having done nothing visible. (It does not even ask for a file to list.) If the file is saved with a starting address of 0100H, the resulting .COM program does execute. It lists the file, and returns to ask for another file when the first file is completed. However, it does not operate properly: with each line it prints a set of characters that do not belong to the file, the same set of characters for each line. If loaded and saved under CP/M 2.2 it operates properly, even when moved to a CP/M Plus diskette. However, if the .REL file is loaded by L80 with the /G option the program operates properly, even under CP/M Plus.

If the .REL file is loaded by Digital Research's LINK, the resulting .COM program executes and lists the file properly, but hangs without returning to request more input when it has finished listing the file. Recovery requires rebooting the system.

I have written to Digital Research and Microsoft, as well as the vendor from whom I bought the computer, a MAX80, and have not received any assistance. Have you heard about this problem? Or might some of your readers have some information that might be helpful.

Sincerely yours,
Maynard B. Neher
16637 Diaz Drive
San Diego, CA 92128

Tiny BASIC

Dear *DDJ*:

I have enjoyed your journal very much over the years. I especially liked your articles on Tiny BASIC and Small-C. I implemented Tiny-BASIC on the 6800 after your first articles. I wrote a Small-C for the 6809 when you started the Small-C series and did one for the 68000 later. When you published the article on Tiny BASIC for the 68000 in the February issue [*DDJ* #100 February 1985] I couldn't resist downloading the code and trying it.

I have a suggestion for improvement which I would like to pass along. The RND function does not generate very good random numbers. I wrote the enclosed test program which generates pairs of 1 digit numbers. These should be evenly distributed over the 100 possible combinations. If you try the program you will see that there are several pairs of digits that don't appear at all.

I have enclosed an improved RND function which eliminates this problem. It is only a first try and there is room for improvement in two areas: First it is restricted to a limit of 65535 since I didn't bother to implement a 32-bit multiply. Second it is based on the multiplicative algorithm and I didn't spend any effort picking the best multiplier. It should have a sequence length of 4,294,967,296 however.

John P. Byrns
1953 Governors Ln.
Hoffman Estates, IL 60195

DDJ

The Ultimate Programmer's Editor

WENDIN'S XTC™

VERSION 3.0 STILL ONLY $99⁰⁰

SUPER PROGRAMMERS edit in XTC to make software development a snap! Just look at these powerful features:

MULTITASKING

XTC's built-in multitasking lets you run your macros in the foreground or independently in the background while you continue editing. A background process has full access to editor resources, and can be used to translate code from one language to another in REAL TIME, print files in the background, or even scan syntax while you type in code. Best of all, you can use XTC to edit source and documentation in any programming language!

COMPILE IN WINDOWS

All DOS compilers and utilities can be executed from within XTC using a single keystroke. While it runs, XTC captures your compiler's output and redirects it into your text, so you can compare compiler messages with your source code ON THE SAME SCREEN. And using XTC's macro language, Turbo Pascal is literally only a keystroke away. You can use other compilers and utilities inside XTC too — like Lattice "C," Microsoft Pascal, and IBM's Basic, to name a few.

MACRO LANGUAGE

XTC has the most powerful macro language in the editing world. XTC's macros aren't just keystrokes assigned to keys; they're real programs that can be used to automatically edit source code and data files. Like any real programming language, XTC has control structures like IF THEN ELSE, WHILE DO, REPEAT UNTIL, FOR NEXT, DUPLICATE N TIMES, INDEFINITE LOOP, EXIT, and BREAK LOOP. XTC also has INTEGER, BOOLEAN, and STRING variables to hold numbers, conditions, and pieces of text.

WINDOWS & BUFFERS

With XTC you can display up to 8 different files or parts of the same file on the screen at once. XTC's windows are programmable and can even be linked together to share files. XTC also has 20 other buffers that you can use to hold files and blocks of text.

WORDSTAR COMPATIBILITY

If you already know Wordstar commands, then you don't need to learn a new set of commands. If you want to customize XTC, just write macros to emulate the key layout you're used to. XTC can also read Wordstar files, and can even strip off all of the non-standard high bits with a single command.

LARGE FILE EDITING

XTC lets you edit files entirely in memory (using all available memory), or paged to disk, for maximum flexibility. You can choose how XTC buffers text.

INCREDIBLE EXTRAS!

- UNDO N TIMES
- REMOVE WORDSTAR HIGH BITS
- EDIT GRAPHICS DISPLAYS
- AUTOINDENTING MODE
- TAB EXPANSION/ COMPRESSION MODE
- EXTRA LONG LINES
- MACRO COMPILER
- TELEPHONE SUPPORT
- 150 PAGES OF DOCUMENTATION
- RUNS ON IBM / PC, XT, AND / AT COMPUTERS (AND TRUE COMPATIBLES)

COMPLETE SOURCE CODE

XTC comes with 7,000 lines of source code jam-packed onto two DSDD disks. Includes 13 modules written in Pascal, and 2 assembly libraries you can use to access the PC's screen, intercept software interrupts (like INT 21H functions), allocate and deallocate memory, and load and execute programs. It's all included FREE for your recreation and enjoyment!

XTC outperforms any other programmable editor on all IBM/PC, /XT, and /AT computers (and true compatibles). If you want to feel the power of XTC before you buy it, just ask for our demo disk (only $10) and try it out. When you buy XTC, we'll knock 10 bucks off the price.

To get your copy of XTC now, order it over the phone — we can ship it the same day you call! Or, send in an order, just like this one:

XTC	**$99.00**
Macro Compiler	**FREE**
Shipping, Handling, Insurance	**3.50**
Want it COD? Add this	**1.90**

TOTAL IT UP, AND SEND IT QUICK!

WENDIN™

BOX 266
CHENEY, WA 99004

The people who make quality software tools affordable.

INCLUDED FREE!
Macro Compiler with full source code in "C"! Translates Ada control statements into XTC macros. It even runs in XTC windows!

ORDER HOTLINE
509/235-8088
CREDIT CARDS WELCOME!

Ada® is a registered trademark of the U.S. Department of Defense. Turbo Pascal is a trademark of Borland, Inc. XTC is a trademark of Wendin, Inc.

Circle **no. 112** on reader service card.

DR. DOBB'S CLINIC

by D. E. Cortesi

Random Moves

We are still collating the letters we've received on that pseudo-random number generator (PRNG) we showed in February. Don't use it. We promise emended and improved versions in Forth and assembly language next month.

Turbo and CP/M Plus

The estimable Turbo Pascal product has a design error that makes its 8-bit version unusable under CP/M Plus (CP/M 3.0) and dubious under CP/M 2.2. When you compile a program to form a .com file, Turbo includes several kilobytes of runtime support in the output, along with the code generated from your Pascal source. That's not the problem; runtime support code is necessary.

Part of the job of that runtime library is to initialize the program's environment when it is loaded and begins execution. Anyone who has written assembly code for CP/M knows what must be done: the stack pointer must be set up and the size of available storage discovered. The usual way of sizing storage is to load the BDOS entry address from location 6.

It's a universal CP/M convention that system programs can reserve storage for themselves at the top of the program area by reducing that address. DDT and SID do it; XSUB does it; spoolers and key-macro programs do it; and CP/M Plus has elevated it from an informal convention to an elaborate system of Resident System Extensions (RSXs). As a result, the top of storage is a dynamic value in CP/M. It can change from day to day if the BIOS expands and from moment to moment as system programs are loaded.

The Turbo compiler, however, treats the size of storage as a constant. At compile time, it places in the runtime code a constant number reflecting the size of the program as the compiler found it; when the .com is executed, it expects storage to be exactly the same size. If programs compiled under Turbo V.3 find less storage than the compiler had, they abort with the message "Not enough memory." This has nothing to do with how many bytes the program needs. Indeed, our test program didn't need any; it simply wrote "hello" 10 times. If we compile it when no RSXs are present but run it when one is, it aborts. Run under SID, it aborts. It's a strange oversight in an otherwise solid product.

Turbo and Standard Pascal

Understand, please, that we *like* Borland and its marketing policies. Its products give good value for the money, and Borland backs them with support that is much better than average. Between Borland and the "shareware" pioneers, a whole new level of price/performance is being established in the software industry. So spare us the letters about how Borland has done so much for software, etc., etc. We appreciate all that.

Turbo Not Standard

But we are getting just a tad tired of the claim that Borland's Turbo Pascal is "standard." It is definitely *not* standard Pascal, which, in a compiler that seems destined to dominate the micro world, is a damn shame. Its price and good user interface commend it for school use, but we certainly couldn't recommend it for that purpose, and we doubt any college computer science department would accept it despite its friendliness.

Let's talk about the Pascal standard, how Turbo violates it, and what that costs its users.

The Ignored Standard

There *is* a Pascal standard. In the U.S., it is sanctioned by the IEEE Computer Society and the American National Standards Institute (ANSI); abroad, by the International Standards Organization (ISO). Although ANSI finalized it in December 1982, it was available at least two years earlier in drafts that differed from each other in only the minutest points and that nobody expected would change in any significant way prior to final approval.

The standard language omits features that many people want. Although it defines a minimum Pascal, it explicitly opens the door to extensions. It defines an extension as "a modification to . . . this standard that does not invalidate any program complying with this standard . . . except by prohibiting the use of one or more particular spellings of identifiers." In other words, you may add any features you like to Pascal as long as you compile standard programs correctly. The last clause even lets you add new reserved words to the language, words that could be user-defined names under the standard.

The standard lays two burdens on the developer. First, you must document variations from the standard in certain ways; that's usually no problem. The other is that a translator must "be able to determine whether or not a program violates . . . this standard . . . and report the result of this determination to the user." In other words, you must warn the user when a program is not standard Pascal. That's essential to portability; otherwise what you intend to be a standard (hence, portable) program can slip

into an unstandard state that is undetectable until you attempt to port it.

Standard Pascal is a small language but still bigger than, say, Fortran IV; more flexible than, say, COBOL; and easier to teach, to use, and to implement than, say, Ada. If there were general compliance with the standard, we could teach people a single language they could use anywhere, at least for small and medium problems. If you moved to a new implementation, you would have to learn the peculiar extensions it supported, but you could still do useful programming while you learned. And programs you wrote using the standard language would be portable anywhere.

It's ironic that many newer and less forgiving standards (e.g., for graphics, for terminal escape sequences, for various buses) garner wide respect and conformance, while the modest and unconfining Pascal standard, draws only scorn from developers.

Turbo Versus the Standard

The Turbo manual (*Turbo Pascal Version 3.0 Reference Manual*, Borland International, Inc., 1985) doesn't contain the kinds of documentation called for by the standard, but it does have Appendix D, "Turbo vs. Standard Pascal" (p. 319). This shows at once that Borland is unaware of the ANSI/ISO standard. "The Turbo Pascal language," it says, "follows the Standard Pascal defined by Jensen & Wirth in their *User Manual and Report* . . . ."

We hate to disturb such blissful slumber, but the language has changed in the 10 years since the *User Manual and Report* was published. The ANSI/IEEE committee convened in 1979 and published its first draft standard in 1980, two years before Turbo was born.

Turbo follows its outdated standard "with only minor differences introduced for the sheer purpose of efficiency." Efficiency for whom or what? For users who want to port programs or to port their hard-won knowledge of the language? Surely not! What then? Efficiency in the execution of the generated object programs? We will examine the case for that as we go; it doesn't hold up.

Could the changes be to improve the efficiency of the compiler itself? Very possibly; we'll see that the elimination of some hard cases may have let the compiler be a little simpler.

Or could the manual be referring to the efficiency with which the developers can bring a product to market? Did Turbo drop the hard parts so the developers could finish their compiler more quickly?

New and Dispose

Turbo v.1.0 skipped the standard way of managing dynamic storage, the procedures New and Dispose. V.2 added them but in incomplete form. Turbo supports the basic features: New(*ptr*) creates a new one of whatever *ptr* is declared to address, and Dispose(*ptr*) reclaims the space used by whatever *ptr* points to.

The standard allows a special case for allocating variant records (records that may have different lengths depending on a tag field). Suppose that *ptr* bases a variant record type and that *tag* is one of the constants that select among the variants. New(*ptr*) will allocate space to hold the longest variant, but New(*ptr,tag*) should allocate just enough space to hold the *tag* variant. Dispose(*ptr,tag*) is then required to release such a record.

Turbo omits these special forms. Why? Not to enhance the user's efficiency; it will seriously complicate your life if you try to import to Turbo a program that uses this feature. Nor for the program's efficiency; the shortest variant of a record is often the most common one, and lacking a way to allocate short records, you can waste a lot of dynamic storage—sometimes enough to make a program infeasible.

Turbo didn't have to ban these cases. It could have accepted them while ignoring the *tag* parameters. Then standard programs would at least compile. But why not support them in full? The standard says *tag* must be a constant; therefore, Turbo could still tell at compile time how many bytes New allocates or Dispose releases. The extra compiler code couldn't be large, and the cases have no effect on the runtime library.

But the manual adds insult to injury. "The restriction," it chirps, "is easily circumvented by using the standard procedure GetMem." Wrong on every count! GetMem is not "standard"; that name is unique to Turbo. Nor is GetMem type-secure, as New is. Plus it requires a count of bytes to allocate, so to use it you must know in detail how Turbo allocates space in variant records. Then heaven help you if you change the record type but forget to change the GetMem calls! (Turbo has a nonstandard SizeOf function, but it won't give you the size of a particular record variant.)

The Page Procedure

The standard specifies a list of procedures and functions that a compiler must predefine. Turbo has all the trigonometric functions, all but two of the I/O procedures (discussed below), and dozens of nonstandard routines for "turtle" graphics, special I/O operations, DOS calls, and so on.

But it lacks the simple procedure Page. Page(f), where f is a text (ASCII) file, is supposed to append a newline if the file isn't currently at the head of a line then "cause an implementation-defined effect on the textfile f, such that subsequent text . . . will be on a new page if the textfile is printed on a suitable device." Like the other Pascal I/O routines, Page with no parameter works on the file output by default.

Page may not seem like much of a loss, but its absence puts the user in an infuriating bind. Like the other Pascal I/O procedures, Page cannot be duplicated by user code. User procedures can't have default parameters, and there is no way to perform the essential test of a file variable to see if it is at the head of a line. (A Page that always writes a newline will sometimes force an extra blank sheet where the standard Page will not.)

What do you do to import a standard program that relies on Page? You find every use of Page and change it to call one of two procedures that you must write yourself—procedures that can't duplicate the standard actions perfectly. If the output isn't satisfactory, you must dig into the program's logic. Changing the logic of an imported program, one written perhaps by total strangers, because of a stupid omission from a compiler is not a good use of time. If you write a program that you will later export, you must include your homebrew PageNamedFile and PageDefault procedures, surrounded with big comment blocks explaining how to alter them to use Page if it should be available.

What would Page cost: 64 bytes of code in the runtime library and maybe 32 bytes of table space in the compiler? Its overhead would be undetectable in a compile and zero at runtime. Why did Turbo omit it? We'll never know. Appendix D says it's because "the CP/M operating system does not define a form-feed character," but that is pure waffle. If there is one thing common to all CP/M systems, it's ASCII, whose form feed is respected by every printer we know.

Get and Put

Wirth gave a mathematician's definition of I/O. He began with a rigorous definition of the file data type. Then he chose a minimal set of operators on that type, operators whose actions are regular enough to be useful in formal verifications of program correctness. Finally he defined all file operations in terms of these minimal operators. The standard follows the *User Manual and Report* in giving a rigorous logician's definition of file I/O, all based on the operators Get and Put.

What do these fundamental operations do? First, understand that to Wirth a file variable is a species of pointer variable. He says that a file is a sequence of objects, while a file variable is a pointer to only one of the objects in the sequence. The declaration

```
fr : file of real;
```

asserts two things: somewhere there is a sequence of real numbers (possibly empty), and under the right circumstances *fr* will address one of them. The statement

```
Reset(fr);
```

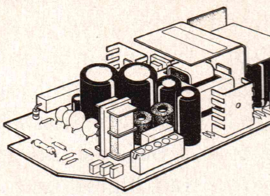
sets up the circumstances; *fr* now ad-

ALL SALES ARE MADE SUBJECT TO THE TERMS OF OUR 90 DAY LIMITED WARRANTY. A COPY OF THIS WARRANTY IS AVAILABLE FREE, ON REQUEST.

DIGITAL RESEARCH COMPUTERS
(214) 225-2309

Switching Power Supply!

+ 5VDC - 8 Amps
+12VDC - 5 Amps
-12VDC - 1 Amp
(81 WATTS MAX)



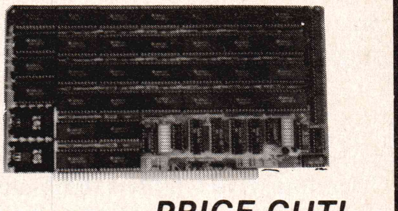
$29.95 ea. | 4 FOR $99 | ADD $2 EA. UPS

BRAND NEW UNITS, MFG. BY BOSCHERT FOR HEWLETT PACKARD! PERFECT FOR SMALL COMPUTER AND DISK DRIVE APPLICATIONS #XL81-5630. INPUT 110V/220V, 50/60 HZ. NOMINAL OUTPUTS: +5VDC @ 8A, +12VDC @ 5A, -12VDC @ 1A. TOTAL MAX OUTPUT. MUST BE LIMITED TO 81 WATTS TOTAL! (WITH PIN OUT SHEET.) ORIGINAL FACTORY BOXED.

64K S100 STATIC RAM

$139.00 KIT

NEW!
LOW POWER!
150 NS ADD $10



BLANK PC BOARD WITH DOCUMENTATION $49.95

SUPPORT ICs + CAPS $17.50

FULL SOCKET SET $14.50

FULLY SUPPORTS THE NEW IEEE 696 S100 STANDARD (AS PROPOSED)

FOR 56K KIT $125

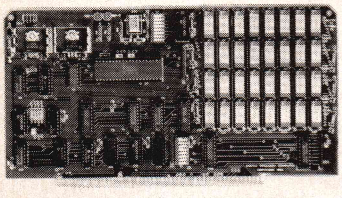
ASSEMBLED AND TESTED ADD $50

FEATURES: *PRICE CUT!*
* Uses new 2K x 8 (TMM 2016 or HM 6116) RAMs.
* Fully supports IEEE 696 24 BIT Extended Addressing.
* 64K draws only approximately 500 MA.
* 200 NS RAMs are standard. (TOSHIBA makes TMM 2016s as fast as 100 NS. FOR YOUR HIGH SPEED APPLICATIONS.)
* SUPPORTS PHANTOM (BOTH LOWER 32K AND ENTIRE BOARD).
* 2716 EPROMs may be installed in any of top 48K.
* Any of the top 8K (E000 H AND ABOVE) may be disabled to provide windows to eliminate any possible conflicts with your system monitor, disk controller, etc.
* Perfect for small systems since BOTH RAM and EPROM may co-exist on the same board.
* BOARD may be partially populated as 56K.

256K S-100 SOLID STATE DISK SIMULATOR!

WE CALL THIS BOARD THE "LIGHT-SPEED-100" BECAUSE IT OFFERS AN ASTOUNDING INCREASE IN YOUR COMPUTER'S PERFORMANCE WHEN COMPARED TO A MECHANICAL FLOPPY DISK DRIVE.

PRICE CUT!



FEATURES:
* 256K on board, using + 5V 64K DRAMS.
* Uses new Intel 8203-1 LSI Memory Controller.
* Requires only 4 Dip Switch Selectable I/O Ports.
* Runs on 8080 or Z80 S100 machines.
* Up to 8 LS-100 boards can be run together for 2 Meg. of On Line Solid State Disk Storage.
* Provisions for Battery back-up.
* Software to mate the LS-100 to your CP/M* 2.2 DOS is supplied.
* The LS-100 provides an increase in speed of up to 7 to 10 times on Disk Intensive Software.
* Compare our price! You could pay up to 3 times as much for similar boards.

BLANK PCB (WITH CP/M* 2.2 PATCHES AND INSTALL PROGRAM ON DISKETTE) $69.95 (8203-1 INTEL $29.95)

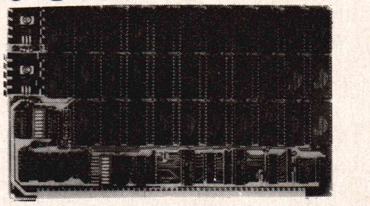
(ADD $50 FOR A&T) #LS-100

$169.00 (FULL 256K KIT)

64K SS-50 STATIC RAM

$119.00 (48K KIT)

NEW!
LOW POWER!
RAM OR EPROM!



BLANK PC BOARD WITH DOCUMENTATION $52

SUPPORT ICs + CAPS $18.00

FULL SOCKET SET $15.00

56K Kit $129

64K Kit $139

ASSEMBLED AND TESTED ADD $50

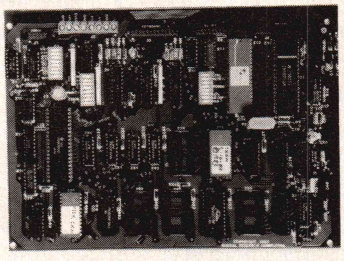
FEATURES:
* Uses new 2K x 8 (TMM 2016 or HM 6116) RAMs.
* Fully supports Extended Addressing.
* 64K draws only approximately 500 MA.
* 200 NS RAMs are standard. (TOSHIBA makes TMM 2016s as fast as 100 NS. FOR YOUR HIGH SPEED APPLICATIONS.)
* Board is configured as 3-16K blocks and 8-2K blocks (within any 64K block) for maximum flexibility.
* 2716 EPROMs may be installed anywhere on Board.
* Top 16K may be disabled in 2K blocks to avoid any I/O conflicts.
* One Board supports both RAM and EPROM.
* RAM supports 2MHZ operation at no extra charge!
* Board may be partially populated in 16K increments.

THE NEW ZRT-80 CRT TERMINAL BOARD!

A LOW COST Z-80 BASED SINGLE BOARD THAT ONLY NEEDS AN ASCII KEYBOARD, POWER SUPPLY, AND VIDEO MONITOR TO MAKE A COMPLETE CRT TERMINAL. USE AS A COMPUTER CONSOLE, OR WITH A MODEM FOR USE WITH ANY OF THE PHONE-LINE COMPUTER SERVICES.

FEATURES:
* Uses a Z80A and 6845 CRT Controller for powerful video capabilities.
* RS232 at 16 BAUD Rates from 75 to 19,200.
* 24 x 80 standard format (60 Hz).
* Optional formats from 24 x 80 (50 Hz) to 64 lines x 96 characters (60 Hz).
* Higher density formats require up to 3 additional 2K x 8 6116 RAMS.
* Uses N.S. INS 8250 BAUD Rate Gen. and USART combo IC.
* 3 Terminal Emulation Modes which are Dip Switch selectable. These include the LSI-ADM3A, the Heath H-19, and the Beehive.
* Composite or Split Video.
* Any polarity of video or sync.
* Inverse Video Capability.
* Small Size: 6.5 x 9 inches.
* Upper & lower case with descenders.
* 7 x 9 Character Matrix.
* Requires Par. ASCII keyboard.

FOR 8 IN. SOURCE DISK (CP/M COMPATIBLE) ADD $10



$89.95 #ZRT-80
(COMPLETE KIT, 2K VIDEO RAM)

BLANK PCB WITH 2716 CHAR. ROM. 2732 MON. ROM
$49.95
SOURCE DISKETTE - ADD $10
SET OF 2 CRYSTALS - ADD $7.50

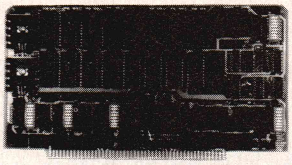
32K S100 EPROM/STATIC RAM

FOUR FUNCTION BOARD!

NEW! NEW!

EPROM II FULL EPROM KIT $69.95

A&T EPROM ADD $35.00



BLANK PC BOARD WITH DATA $39.95

SUPPORT IC'S PLUS CAPS $16

FULL SOCKET SET $15

We took our very popular 32K S100 EPROM Card and added additional logic to create a more versatile EPROM/RAM Board.

PRICES SLASHED!

FEATURES:
* This one board can be used in any one of four ways:
  A. As a 32K 2716 EPROM Board
  B. As a 32K 2732 EPROM Board (Using Every Other Socket)
  C. As a mixed 32K 2716 EPROM/2K x 8 RAM Board
  D. As a 32K Static RAM Board
* Uses New 2K x 8 (TMM2016 or HM6116) RAM's
* Fully Supports IEEE 696 Buss Standard (As Proposed)
* Supports 24 Bit Extended Adressing
* 200 NS (FAST!) RAM'S are standard on the RAM Kit
* Supports both Cromemco and North Star Bank Select
* Supports Phantom
* On Board wait State Generator
* Every 2K Block may be disabled
* Addressed as two separate 16K Blocks on any 64K Boundary
* Perfect for MP/M* Systems
* RAM Kit is very low power (300 MA typical)

32K STATIC RAM KIT — $109.95
For RAM Kit A&T — Add $40

Digital Research Computers

P.O. BOX 461565 • GARLAND, TEXAS 75046 • (214) 225-2309

TERMS: Add $3.00 postage. We pay balance. Orders under $15 add 75¢ handling. No. C.O.D. We accept Visa and MasterCharge. Tex. Res. add 5-1/8% Tax. Foreign orders (except Canada) add 20% P & H. Orders over $50, add 85¢ for for insurance.

*TM OF DIGITAL RESEARCH INC. (CALIF.)

WE ARE NOT ASSOCIATED WITH DIGITAL RESEARCH INC. (CALIF.) THE SUPPLIERS OF CPM SOFTWARE

Circle no. 33 on reader service card.

dresses the first real number in the file. For comparison, recall that

```
pr : ^real;
```

sets up a variable *pr* that points to a single real number, as in

```
x := 3.5 * pr^;
```

Therefore, Reset(*fr*) sets things up so that *fr* addresses a real number:

```
x := 3.5 * fr^;
```

Thus did Wirth unify the treatments of I/O and dynamic storage.

Standard Get and Put

The Get procedure advances a file one unit in its sequence. After Get(*fr*), *fr* addresses the next real number in the sequence.

Rewrite opens a file for output; the statements

```
nr : file of real;
Rewrite(nr);
```

create an empty sequence of real numbers and make *nr* point to a real of undefined value. You may use *nr* exactly like any other pointer to reals:

```
nr^ := sqr(pr^) + sqr(fr^);
pr^ := sqrt(nr^);
```

The statement

```
Put(nr);
```

appends the current referent of *nr* to its file and makes *nr* point to undefined space again.

Do you find these ideas confusing, difficult, baffling? Do you think they make for an inflexible system? Or do you agree that they are simple, logical, consistent, and highly useful?

The Borland programmers did not. Turbo does not support Get and Put, nor does it permit a file variable to be used as a pointer. It offers only the Read and Write procedures (which to Wirth are composite operations defined in terms of Get and Put).

Simplicity?

Appendix D offers four excuses for the omission. One is that Read and Write are "easier to understand." A computer science teacher would never say that. Good teachers explain the basis of things. You cannot explain the basis of Read and Write without referring to Get and Put; you can state their actions as arbitrary rules to learn by rote, but you can't explain them.

Of course, if you are already an experienced programmer—one used to Fortran or BASIC, perhaps—you might find the Read and Write procedures more *familiar* than Get and Put, and you might confuse familiarity with simplicity. That's a common error, as anyone will attest who has tried to explain the arbitrary, but totally familiar, rules of English to a foreigner.

Versatility?

One of the three other excuses is that Read and Write are "far more versatile" than Get and Put. That's rubbish, but our hypothetical ex-Fortran user wouldn't realize it without close study of Pascal.

With Put, you can assign and reassign an output value as often as you like; the value isn't committed to the file until Put is issued. That concept isn't in the thinking of a PL/I user; in most languages, once you associate data with a file, it's gone. Once you comprehend the idea, however, you can find lots of uses for *writing* data without *putting* it.

Get is even more useful. If you've read Kernighan and Plauger's *Software Tools*, you may recall how often they resort to the ungetc( ) procedure to replace the character last read from a file. With Pascal standard input, you can look at the next value coming from a file without committing yourself to removing it from the file. You simply test the value addressed by the file variable. If you want it, you assign it somewhere else and issue Get. You may also leave it for another part of the program to use, or discard it with Get. Here's a procedure, free of side effects, to strip blanks from an input file:

```
procedure fstrip(var f: text);
begin
   while (not eof(f))
      and (' '=f^) do
         Get(f)
end;
```

Afterward, either the next element of the file is a nonblank or the file is finished. Try writing a comparable function in C or BASIC—or in Turbo! It takes either ungetc( ) or your own input function with a static variable.

Because Get and Put are standard and highly useful, you may assume that Pascal programmers make use of them. A lot of Pascal programs are floating around on minis and mainframes and in textbooks. The odds are good that if you try to import one to Turbo, you'll fail because the program uses Get or Put. You may expect to find as well that the assumptions behind Get and Put are woven deeply into the program's logic; most programs will need major changes to run without them.

Less Overhead?

Appendix D offers the excuse that, without Get and Put, "variable space overhead is reduced, as file buffer variables are not required." True, the compiler would have to allocate a buffer the size of one file data item in addition to the sector buffer it needs for DOS I/O. Alternatively, it could round the size of a data item up to a whole number of DOS sectors, add one sector, and allocate a single buffer of that size (this ensures a complete data item will always fit in the buffer).

Banning standard use of the file variable lets the compiler get by with a single sector buffer. Does that reduce "variable space overhead"? Of course not! Now *you* have to define the variables! You need extra variables to hold data items before they are written to the file, instead of merely assigning and reassigning until a Put is done. You need variables to hold items after reading them from the file and before using them in expressions. And if it turns out the items shouldn't have been read, Turbo offers no ungetc( ), so *you* get to define one—again adding code overhead.

Speed?

Appendix D's premier excuse is that "Read and Write give much faster

I/O." Do they? True, a naive implementation of Get might often copy data from a sector buffer to an item buffer. But Read *requires* copying data to a user-defined variable, whereas a smart implementation of Get could avoid moving data at all. Many times when the file data type is smaller than half a sector (the typical case), Get need do no more than increment a pointer, check for end of sector, and return. That's quick!

If a new sector must be read, a Get implementation might have to move part of the last sector from the end of the buffer to its head before reading more data. But that move always involves *less* than a single data item! Under MSDOS, when the file data type is composite (record, set, or array), a clever compiler would push the whole burden onto the DOS. It would allocate no sector buffer at all, just an item buffer, and use "handle" I/O to read or write only as many bytes as needed.

A good implementation of Get and Put might entail 15 percent more code than the present Turbo library, but by moving less data, it might well execute in 25 percent less time.

Procedure Parameters

One of the least-regarded features of standard Pascal is the ability to pass a procedure or function as a parameter. Let's say "routine" instead of "procedure or function." Then the feature sounds simpler: the ability to pass one routine the name of another. Its purpose is to let a routine's behavior be parameterized along with its data. Unfortunately, neither Jensen nor Wirth nor the standards committee could find a good example of its use except "finding the minimum of a function by bisection." Now really, what can an honest programmer say to that but "Gah, Wha?"—meaning, surely you can omit such an abstruse feature from a language without anyone's noticing!

Let's try to suggest some examples to show why it should stay. The first might be a generalized sort that takes, as a parameter, the name of a function that compares two data items and returns True if one is less than or equal to the other. Now define two comparison routines:

```
function Up(a,b:real):Boolean;
begin Up := (a <= b) end;

function Down(a,b:real):Boolean;
begin Down := ( b<=a ) end;
```

The identical generalized routine will sort ascending if we pass it the name of Up and descending if we pass it Down.

Notice that if we make Up and Down take, not the data to be compared, but ordinal indices to the data, and if we also pass a procedure Swap that interchanges data items given their indices, we will have removed all dependencies on data type from the general sort routine. We need only pass it the Up or Down and the Swap that are appropriate to the data type we want to sort.

As another example, consider a situation where a service routine can detect an error, for which the correct recovery action depends on the conditions under which the routine was

called. There are three ways to handle this: (1) You can write a version of the routine for each recovery method; (2) you can pass the routine an error-recovery action number and put a big error-handling case statement in it; or (3) you can pass the routine the name of an error-handling procedure. Each client may then contain a local procedure to do error recovery in its own style. The service routine remains general, while the recovery actions are localized in those parts of the program that establish the need for them.

Turbo Pascal is not alone in banning procedure parameters; it's the most ignored feature of the language. Why? It seems like a fairly simple thing to implement—at first. At runtime, it amounts to just pushing the address of a routine on the stack. At compile time, the compiler must check that the parameter list of the routine that is passed matches the parameter list that the receiving routine declared, which is easier to do than to describe.

The real problem is the implementation of Pascal's rules for the scope of variable names. When procedure parameters are forbidden, the dynamic scope of a name matches its lexical scope; the compiler can tell from the program text exactly what names are accessible at any point in the program. When procedure parameters are allowed, cases occur where the referent of a name can be resolved only by extra stack linkages. That complicates the runtime code for stack management and for procedure call and exit.

Although Borland may seem to have made a good tradeoff in dropping procedure parameters, the usual arguments can be made for the feature: it's useful and likely to appear in published programs, especially in the more sophisticated ones, those that are at once most valuable and hardest to modify. We might add that, inasmuch as Turbo threatens to be the single most common implementation of Pascal ever, Borland has an obligation to support the whole language.

The stack management is not that complicated anyway. Full Pascal stack linkage no doubt would slow down an 8080 or Z80 implementation (for once there's a legitimate reason for restricting the CP/M version!), but that is not true of the 8086.

What would be the penalty for full Pascal in the MSDOS version of Turbo? The compiler would be slightly more complicated, as noted. And the runtime code? Studies have shown that the overwhelming majority of references are to local and global names and to parameters. The troublesome references, to names defined in intermediate scopes, are very rare. (It's tempting to modify the scope rules to forbid just these references, but that would create incredibly subtle portability bugs.) If the generated code optimizes common usages and penalizes the rare ones, the cost should be small.

Summary

The Pascal standard defines a nice little language, but despite the language's popularity and the standard's age, the features of the language and the liberality of the standard are still misunderstood or unappreciated. The justifications in the Turbo manual's Appendix D just don't hold water; likely the real reason Turbo is nonstandard is that the Borland programmers were in a hurry to get to market so they dropped some features that were obscure (to them) or of little use (to people who don't understand Pascal well). Yet Turbo Pascal has many admirable features. If it added to them full compliance with the Pascal standard, it would be without question the finest implementation of Pascal for personal computers and among the finest on any computer.

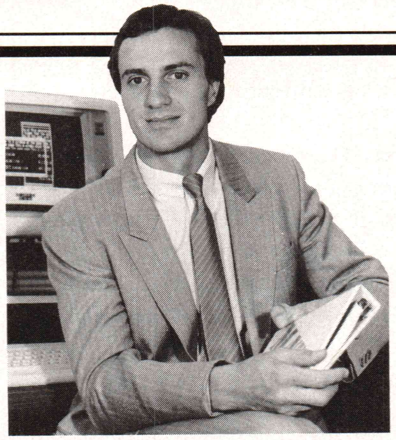
DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 190**.

We're getting hardnosed at Softway.

From now on MATIS™ is only $49⁹⁵!

(MATIS, the complete User Interface development tool has been selling for $150.)



Denis Moran
President, Softway, Inc.

Why the radical price cut?

We decided after looking over the competition that MATIS had so many advantages it should be made available to more programmers. We decided to compete aggressively so you could easily afford to have MATIS in **your** bag of tricks. We hear from MATIS users in the USA and France that it is a truly loveable product. Sooo...we're running this big ad to promote our new low price.

MATIS windows are beautiful.

Display any portion of any screens you create at any point in your program. Scroll in any direction manually with cursor keys...or automatically.

And the screens are HUMUNGEOUS!

MATIS screens can be just about as big as you want...up to 65,534 rows by 65,534 columns! The number of screens is only limited by available memory.

Print big MATIS screens directly.

One command sends your screens to your printer with no need to program special routines when your virtual screen is bigger than your terminal screen.

User input fields are a snap.

Creating fields for data entry is easy with no limit to size or number by screen. Request for input separately or in blocks.

Control your keyboard with MATIS.

It keeps track of keys that are pressed during the execution of your program and lets you assign specific functions to selected keys.

Control that screen too!

MATIS is extremely versatile and flexible when it comes to controlling lines, columns, fields, and text. They can be modified, transferred, displayed or moved with a single command. All video attributes are supported: color, reverse video, blinking...you name it, you got it.

Want an interactive screen builder?

You've got it with MATIS. It's called "MATPAGE"™ and it lets you create and modify any of your screens in an interactive mode.

MATIS adds over 70 routines to your program.

Written in Assembler, MATIS routines are fast and powerful giving your program improved efficiency and enhanced visual appeal, while they reduce its size and maintenance worries. And MATIS separates screen design from the core of your program.

MATIS is unique.

We don't think there's a single program that combines as many tools in one package as completely or as well as MATIS. It interfaces with Intepreted and Compiled BASIC (Microsoft), C (Lattice, Microsoft, Aztec), PASCAL (IBM, Microsoft) and ASSEMBLER. All you need is an IBM* PC/XT or true compatible under DOS, 128k or RAM, monochrome or color monitor.

You get an easy to follow no-frills manual and a 30-Day Money Back Guarantee.

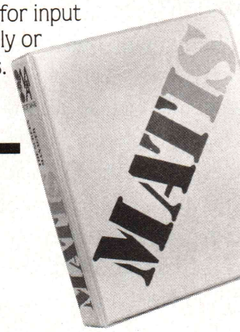
Late News: MATIS/T™ for TURBO-PASCAL** only $29.95

An indispensable add-on at a dynamite price. What more can we say?

Denis Moran

Denis Moran

™ MATIS, MATIS/T, & MATPAGE are Trademarks of Softway, Inc.
*IBM is a Reg. Trademark of IBM Corp.
**Turbo Pascal is a Reg. Trademark of Borland International



Softway, Inc.

500 Sutter Street, Suite 222 F
San Francisco, California 94102
(415) 397-4666

24-Hour Credit Card Orders By Phone:
1 (800) 227-2400 EXT 989 In California: 1(800) 772-2666 EXT 989

Please ship the following at once. I understand there is a 30-day money back guarantee.

_____ Copies of MATIS at $49.95 plus $3 shpg. $ _____
_____ Copies of MATIS/T for TURBO PASCAL at $29.95 plus $2 shpg. $ _____
California residents, add 6½% sales tax $ _____
☐ I like to read specs, so send me a folder. Total $ _____

Name _____
Address (please no P.O. Box) _____
City/State/ZIP _____
Phone ( ) _____ Signature _____
Make payment by money order, check, or charge card ☐ VISA ☐ MASTER CARD
Number _____ Exp. _____

Distributed in Europe By: MICRO APPLICATION SOFTWARE • 147 Avenue Paul Doumer, 92500 RUEIL MALMAISON FRANCE, Tel (1) 732.92.54

Circle **no. 92** on reader service card.

by Allen Holub

As part of my ongoing struggle to turn MSDOS into Unix, I write a lot of Unix-like utilities. This month we'll look at a better MSDOS directory utility called "ls" (after the Unix model). Ls uses several subroutines useful in their own right. A multicolumn print utility and an MSDOS interfacing utility (which corrects various deficiencies in the one supplied by Lattice) are two of these.

How To Use Ls

Ls prints a sorted directory in a multicolumn format. The columns are read like a magazine; that is, an entire column is read from top to bottom, then the next column is read. Ls prints volume labels, which are in boldface, first. Next it prints directories; they are underlined. Then it prints the files. Finally, it prints the amount of space used by the directory (instead of the space left on the disk), along with a file and directory count. The figure (page 21) shows various sample outputs from ls.

Usage is: ls [options] <file list>. The legal command line options are:

-a
List all files, including hidden files. Like Unix, all files whose names begin with a period (.) are considered invisible (even if the hidden attribute bit isn't set); -a causes these files to be printed too.

-c<num>
Print the output in <num> columns instead of the default five. Up to six columns can fit on a screen. The command

```
ls -c1 *.c >foo.bat
```

is useful for making batch files. No spaces are allowed between -c and the number.

-d
List only directories (no files).

-e
Sort by extension and then by filename. This groups all the .c files together and all the .obj files, and so on. The filenames are sorted within their extension categories (i.e., bar.c precedes foo.c, and both precede bar.obj).

-f
List only files (no directories).

-l
List in long format. -l lists the file size, the create time and date, and the file's attributes. Possible attributes are:

D – (D)irectory
H – (H)idden
L – Volume (L)abel
M – File has been (M)odified since last backup
R – (R)ead only
S – (S)ystem

-s
Suppress directory name-underlining and label-boldfacing so that you can direct the output to a printer without the extra escape sequences. Note that directories sort to the top of the list because the graphics require an ESC (which has a smaller value than any letter) at the beginning of the name string. Because the ESC is no longer present, the directory names will be mixed in with the filenames when you use -s.

-u
Print directory unsorted.

You may combine options (ls -ac3s *.c) or list them separately (ls -a -c3 -s *.c). You can place them anywhere on the command line (ls *.c -l is the same as ls -l *.c).

The <file list> is a list of the files you're looking for. You can list files explicitly (ls foo.c bar.c) or with ambiguous references (ls *.c). A directory may precede the filename (ls \src\tools\ls.c). You can use / as well as \ to separate directory names (ls /src/tools/ls.c). If you request a single directory, then ls lists the contents of that directory. If no <file list> is given, ls prints all files in the current directory.

To make the underlining and boldfacing work, you must find the line "device=ANSI.SYS" in your config.sys file when the system boots. If you don't do this, your screen will look funny unless you use the -s option.

Getting a Directory

You can access MSDOS directories in two ways. A set of low-level DOS function calls "Search for First Entry" (0x11) and "Search for Next Entry" (0x12) work much like the equivalent CP/M functions. (An example of CP/M directory accessing appeared in the March 1985 C Chest.) Because functions 11 and 12 force you to construct an FCB, and they search only the current directory, they're pretty hard to use.

Luckily, DOS versions 2 and higher provide two easier-to-use functions: "Find First" (0x4e) and "Find Next" (0x4f). When you pass Find First a pointer to a string holding a filename, it loads information about the requested file into the current disk transfer area (DTA). If you give an ambiguous file reference, subsequent files that match the reference are retrieved with a series of Find Next calls. Find Next must use the same DTA as Find First used.

If an error is encountered (such as not finding the requested file), the carry bit is set when the DOS interrupt returns, and an error code is put into the AX register. DOS v.2 sets the AX register to 0 on success, but DOS v.3 doesn't seem to do this; you actually have to look at the carry bit.

These two routines have several nice features. The name can contain ambiguous file references and directory specifiers, so you can search for a file anywhere in the file system. You can even use constructs like ". ." or ". .\. ." if you want. Perhaps nicer, you can use / as well as \ to separate directory names in a complete file reference. (This is actually true for all the low-level MSDOS file functions. The insistence on that idiotic backslash is an anomaly of the MSDOS-supplied shell, command-.com. Why couldn't they use / to separate directories and - for command line switches?) Another nice feature is that you can specify a disk id as part of the filename (c:/foo).

All is not wonderful, though. The two directory-searching routines do have problems. Unix directories are just files. The only difference between a Unix directory and any other file is the value of the directory attribute bit. MSDOS uses a similar mechanism, except that directories are special: they can't be accessed like other files, they have a length of zero, and so forth. However, they will show up in a directory search as if they were files (with an attribute bit that says you're looking at a directory).

So, if you request a directory from Find First, you'll get a single listing with the matching directory name. You won't get a listing of the files in that directory; to do this, you have to specify /*.* after the directory name. A search for ". ." will return a structure that actually contains the name of the parent directory.

A second problem is the root directory. MSDOS doesn't think that the root directory exists. If you request / or \ from Find First, it comes back with a file-not-found error. Similarly, requesting b: produces the same error. You need to ask for a file (or *.*).

Another problem is the volume label. A filename returned by DOS has all the blank padding removed and a period inserted between the name and extension. Unfortunately, DOS also puts the period into a volume label. If a disk is labeled LONGLABEL, the DOS directory search will yield LONGLABE.L as the volume label. Ls doesn't do anything to correct this problem.

The final problem is actually a deficiency in the Lattice C I/O library. (The people at Lattice claim that the next revision of the compiler fixes this, but that doesn't help us now.) To use Find First/Next, you need to mess with the DTA (the place where the directory will end up after the function call). Unfortunately, your disk I/O system uses the DTA. If you're not doing any file I/O, you can put the DTA where you want it (with a DOS function 0x1a call) and then forget about it. *But*, if you're working with files too, you'll have to put the DTA back where it came from to do file accesses.

```
C:\SRC\LS ls/

HARDFILE    GAMES      SRC       AUTOEXEC.BAT
BIN         INCLUDE    TEXT      COMMAND.COM
DOS         LIB        UTIL      CONFIG.SYS

3 files (22172 bytes, 21 K), 8 directories
C:\SRC\LS ls -la/

HARDFILE            0    12-20-84 17:22:18 LM
BIN                 0    12-24-84 15:13:04 D
DOS                 0    12-20-84 17:23:04 D
GAMES               0    12-24-84 15:20:32 D
INCLUDE             0    12-24-84 15:17:56 D
LIB                 0    12-24-84 15:22:26 D
SRC                 0    12-24-84 15:13:54 D
TEXT                0    12-24-84 15:20:58 D
UTIL                0    12-24-84 15:17:58 D
AUTOEXEC.BAT       95     4-09-85 17:42:50 M
COMMAND.COM     22042     8-14-84  8:00:00
CONFIG.SYS         35     1-22-85 22:19:06 M
IBMBIO.COM       8964     7-05-84 15:00:00 RHS
IBMDOS.COM      27920     7-05-84 15:00:00 RHS

3 files (59056 bytes, 58 K), 8 directories
C:\SRC\LS ls -a/text/drdobbs

.       FOO        LS.NR           SAVE.BAT
..      CROOT.NR   Q_BITMAP.NR     SORT.NR

5 files (56197 bytes, 54 K), 3 directories
C:\SRC\LS ls -c1fe/text/drdobbs

SAVE.BAT
CROOT.NR
LS.NR
Q_BITMAP.NR
SORT.NR

5 files (56197 bytes, 54 K)
C:\SRC\LS ls -d/text/drdobbs

FOO

1 directory
```

Figure 1.
Sample Outputs from Ls

DOS function 0x2f returns the current DTA in ES:BX. Unfortunately, the Lattice DOS interface functions (bdos, int86, etc.) won't let you get at the ES returned by the get DTA function. They push all the registers, do the DOS call, then restore the registers, overwriting the ES returned by DOS. Nor will a subsequent segread( ) call return the value you need for ES.

My solution to this problem was to write a more sensible DOS interface, which we'll talk about in a moment. Although ls doesn't need to put the DTA back, an example of how to restore the DTA is given in Listing Five (page 41).

Program Description

Ls itself appears in Listing One (page 25). MAXDIR (line 12) is the maximum number of files that will be listed; change this if you need more than 132 (132 names are 6 columns by 22 lines).

Ls uses two subroutines that have appeared in previous columns: qsort (*DDJ* #102, April 1985) to sort the directory and getargs (*DDJ* #103, May 1985) to parse command line arguments. The argument table for getargs is on lines 27–37 of Listing One (getargs.h on line 2 is used by getargs, as we discussed back in May).

The global variables on lines 19–26 are set by the various command line switches. The variables on lines 39–44 are more general purpose.

Directories are stored in an argv-like array of pointers to character strings: Dirs (line 39). Dirv and Dirc (40–41) operate like argv and argc. Total (line 42) is the total size, in bytes, of all listed files. Numfiles and Numdirs (43–44) are the number of files and directories found, respectively.

Find_first( ) and find_next( ) (lines 47–77) do the DOS calls. They use the DOS interface function given in Listing Three (page 34).

The REGS structure is *not* the structure of the same name defined in the dos.h supplied by Lattice. This version of REGS is defined in mydos.h (Listing Two, page 33, lines 14–23). REGS is actually a functional superset of the Lattice REGS structure (that is, you can use the REGS defined in mydos.h to talk to the Lattice interface functions, but you can't use the Lattice REGS to talk to the dos( ) function in Listing Three).

REGS has fields for all the registers you need to access MSDOS, including the segment registers. It is a union that defines the normal registers so that you can easily access the high, low, or both bytes. The two register images are superimposed (i.e., definitions for byte offsets are superimposed over definitions for word offsets). For example, given a pointer p to a REGS structure, p→x.ax accesses the AX register, p→h.ah gets the high byte (the AH register), and p→h.al gets the low byte (the AL register). You can retrieve only the AX, BX, CX, and DX registers in this way. The others (SI, DI, ES, CS, SS, DS) are all accessed through the x part of the union (p→x.es).

The interface function, dos( ) (Listing Three), is called with dos(regp); where regp is a pointer to a REGS type structure. It returns the status register, as returned by DOS. You can use dos( ) only with the

small and one of the medium (the P) memory models because it assumes a 16-bit pointer.

Dos( ) copies all fields of *regp except x.ss and x.cs into their equivalent registers (lines 59–66 of Listing Three). DOS is invoked with the INT 21 on line 68, then *all* the registers are copied back into *regp. The routine returns with registers restored to their original state (i.e., before the dos( ) call). The function gregs( ) (lines 93–124) gets the register contents but doesn't do a DOS call; it's used for initializing a REGS structure.

Find_first( ) and find_next( ) both return 0 on success or the error code passed back by DOS on failure. When they succeed, the current DTA will be loaded with a shuffled-around version of the directory entry for the requested file. You use the FILE_INFO structure, typedefed in mydos.h (Listing Two, lines 30–39), to access this information. Mydos.h also has a bunch of macros for extracting stuff from packed fields (lines 59–79). With these, you can get at the various fields of the creation time and date, as well as test for specific attributes.

Dirtoa( ) (Listing One, lines 86–135) converts a FILE_INFO structure into an ASCII string (see the figure). The ANSI escape sequences used for underlining and boldfacing are added to the string in dirtoa( ).

The final directory-related routine is fixup_name( ) (Listing One, lines 174–223). This routine takes care of all the problems with accessing the root directory and getting the files inside a specified directory that we discussed earlier. It appends *.* or /*.* onto directory names where necessary.

Add_entry( ) (Listing One, lines 225–253) puts the strings returned from dirtoa( ) into the Dirv array. It also processes the -a and -f flags. If an attribute bit is set when you call find_first( ) or find_next( ), then only those files that have the indicated attributes will be found; the -d option is processed on line 281 using this mechanism. Unfortunately, because there's no way to request files (only directories), -f must be processed by hand in add_entry( ) (lines 241–246).

The only remaining nonobvious function in ls.c is ecmp( ) (lines 147–162), which is used by qsort to do a sort by extension. It skips over to the extension parts of the two strings, compares the extensions, and if they match, goes back and compares the actual filename. Refer to the April C Chest for more information on how qsort works.

Multicolumn Printing

The multicolumn printing is done by ptext( ) (Listing Four, page 37), which prints out an argv-like array of pointers to strings in multicolumn format. Ptext( ) originally was written for a version of the Unix print utility pr and works quite well in that application.

Ptext( )'s calling syntax is:

```
ptext(linec, linev, outfile, numcols,
    colwidth, numrows);
char **linev;
FILE *outfile;
```

Linev is the array of string pointers, linec is the number of pointers in the array, outfile is the stream to which output will be sent, numcols and numrows are the number of rows and columns, respectively, and colwidth is the width of a single column.

Ptext( ) is called in Listing One on line 170. Here num_cols is the number of columns. The column width is 80/Num_cols. The number of rows is computed with (dirc/Num_cols) + (dirc % Num_cols != 0); dirc is the total number of names to be printed. Dirc % Num_cols != 0 evaluates to 1 if Num_cols doesn't divide evenly into dirc and to 0 if it does divide evenly.

Several practical problems having to do with multicolumn printing will vary from application to application. The first problem is a lone carriage return. Some text formatters underline text by printing the text itself, then a carriage return, then a line containing all the underscores. When you have several columns, you can't print the carriage return because that takes you to the left edge of the page rather than the left edge of the current column. The problem is solved in ptext( ) by backspacing to the correct

place. If your printer doesn't support a backspace, you'll have to modify this part of the code (lines 93–103).

The next problem is ESC sequences (which take up no space in the output). Different sequences are composed of different numbers of characters. Rather than try to recognize various escape sequences, ptext( ) just assumes that all sequences use four characters (one for the ESC and three normal characters that follow). This assumption lets us handle the various Set Graphics Rendition (SGR) sequences correctly (we need these for the underlining and boldfacing), but it won't handle the escape sequences needed to change fonts on printers and the like. So, you may need to make this part of ptext( ) more sophisticated (the relevant code is on lines 117–129).

The third problem is with the ANSI.SYS driver provided by IBM (at least with DOS v.3 running on a PC/AT—I'd appreciate someone testing other versions of DOS on other machines and sending me the results). If you try to underline the left-most character on a line, the entire line ends up underlined. If you explicitly turn off underlining, the nonunderlined characters overwrite the underscores, but underscores still extend from the last character you wrote to the physical end of line on the screen (i.e., from the place where you wrote a carriage return to the right edge of the screen). To make matters worse, the underline attribute sticks on, so subsequent lines end up underlined too. I didn't want to rewrite ANSI.SYS, so I kludged a solution to this problem in ptext( ). If "IBM" is #defined at the top of the module, a space will be inserted as the left-most character of every line. I realize this isn't a real solution, but it works everywhere I've needed to use ptext( ).

While we're on the subject of ANSI.SYS, I found another bug a few days ago. The "Erase in Display" and "Erase in Line" leave the cursor in the wrong place. Erase in Display should home the cursor; instead, it leaves the cursor at the left edge of row 2. Similarly, Erase in Line puts the cursor at the left edge of the next line; it shouldn't move the cursor at all. Obviously, the driver is just printing blanks to erase the line, but it should put the cursor back where it belongs when it's done.

The Erase in Line problem is particularly bad when you try to erase the bottom line: the screen scrolls up a line when the cursor is sent past the bottom, so, even if you put the cursor back where it belongs, you lose the top line of the screen. If anyone knows any other bugs in ANSI.SYS (or has fixes for them, maybe a new version of ANSI.SYS), please send them in so I can put them in the column.

This month we talked about directories; next month we'll continue the discussion with a version of the Unix utility "make," which automates the recompilation of modular programs.

C Chest (Text begins on page 20)

Listing One

```
 1: #include <stdio.h>
 2: #include <getargs.h>
 3: #include <mydos.h>

 4: /*-------------------------------------------------------------------------+
 5:  *                   LS.C: An MSDOS directory utility                      |
 6:  *                                                                         |
 7:  *        (c) Copyright 1985, Allen I. Holub. All rights reserved.         |
 8:  *   This program may be reproduced for personal, non-profit use only.     |
 9:  *-------------------------------------------------------------------------+
10:  */

11: #define CARRY    0x01      /* Carry flag mask for an 8086                    */
12: #define MAXDIR   132       /* Largest directory which will be printed        */

13: #define BOLDFACE  "\033[1m"  /* Ansi esc sequence to turn bold face on    */
14: #define UNDERLINE "\033[4m"  /*                  "             underline on */
15: #define ALL_OFF   "\033[0m"  /*                  "           attributes off */
16:
17: #define EOS(s)    while(*s) s++   /* Position s at the end of the string */

18: /*-------------------------------------------------------------------------*/

19: int     Longfmt     = 0;          /* Global variables set by command line */
20: int     Unsorted    = 0;          /* switches.                            */
21: int     Files_only  = 0;
22: int     No_graphics = 0;
23: int     Dirs_only   = 0;
24: int     List_all    = 0;
25: int     Num_cols    = 5;
26: int     Ext_sort    = 0;
```

(Continued on next page)

 (Listing Continued, text begins on page 20)

Listing One

```
ARG Argtab[] =
{
        { 'a', BOOLEAN, &List_all,    "List all files (including hidden)" },
        { 'c', INTEGER, &Num_cols,    "Print output in <num> columns"     },
        { 'd', BOOLEAN, &Dirs_only,   "List only directories"             },
        { 'e', BOOLEAN, &Ext_sort,    "Sort by extension"                 },
        { 'f', BOOLEAN, &Files_only,  "List only files"                   },
        { 'l', BOOLEAN, &Longfmt,     "Print directory in long format"    },
        { 's', BOOLEAN, &No_graphics,"Suppress ANSI graphics chars"       },
        { 'u', BOOLEAN, &Unsorted,    "Print directory unsorted"          },
};

#define TSIZE (sizeof(Argtab)/sizeof(ARG))

static char     *Dirs[MAXDIR]   ;       /* Place to put directory       */
static char     **Dirv   = Dirs ;       /* Like argv for directory      */
static int      Dirc     = 0    ;       /* Like argc for directory      */
static long     Total    = 0L   ;       /* Sum of sizes of all files    */
static int      Numfiles = 0    ;       /* Number of  files   found     */
static int      Numdirs  = 0    ;       /*  "       " directories  "    */
extern char  *malloc();

/*----------------------------------------------------------------------*/

find_first( filespec, attributes, regp )
char      *filespec ;
short     attributes;
REGS      *regp;
{
        /*      Get directory information for the indicated file.
         *      Ambiguous file references are ok but you have to use
         *      find_next to get the rest of the file references.
         *      In this case, The regs structure used by find_first
         *      must be passed to find_next. 0 is returned on success,
         *      otherwise the DOS error code is returned.
         */

        regp->h.ah = (char)  FINDFIRST  ;
        regp->x.dx = (short) filespec   ;
        regp->x.cx = attributes         ;

        return (int)( (dos(regp) & CARRY)  ?  regp->x.ax  :  0 );
}

/*----------------------------------------------------------------------*/

find_next ( regp )
REGS     *regp;
{
        /*      Get the next file in an ambiguous file reference. A
         *      call to this function must be preceded by a
         *      find_first call. The regp argument must be the
         *      same register image used by the find_first call.
         *      0 is returned on success, otherwise the error code
         *      generated by DOS is returned.
         */

        regp->h.ah = FINDNEXT ;
        return (int)(  (dos(regp) & CARRY)  ?  regp->x.ax  :  0 );
}

/*----------------------------------------------------------------------*/

doscall( id, regp )                     /* Do the DOS system call      */
REGS     *regp;                         /* specified by "id", *regp    */
{                                       /* is modified to reflect the */
        regp->h.ah = id ;               /* results of that call        */
```

```
        dos( regp );
}

/*----------------------------------------------------------------------*/
dirtoa( s, p, longfmt )
char            *s;
FILE_INFO       *p;
{
        /* Convert a FILE_INFO structure to an ascii string. In longfmt
         * all information about the file (name, size, date & time,
         * mode) is used. Possible modes are:
         *
         *      R - Read only           H - Hidden
         *      S - System              L - Volume label
         *      D - Directory
         *      M - archive bit is set (file has been modified)
         *
         * If longfmt isn't true, only the name is printed. Maximum
         * string length in long format is 52, in short format is 21
         * Volume label names are printed in bold face (since this will
         * put an ESC in the name the volume label will sort to the front
         * of the list). Directories are printed underlined (these will
         * immediately follow the directories). The number of characters
         * in the string (not including the terminating null) is returned.
         */

        char *startstr = s;
        int  i;

        if( !No_graphics && (IS_LABEL(p) || IS_SUBDIR(p)) )
                sprintf(s, "%s%s%s", IS_LABEL(p) ? BOLDFACE : UNDERLINE,
                                        p->fi_name, ALL_OFF );
        else
                sprintf(s, "%s", p->fi_name );

        EOS(s);

        if( longfmt )
        {
                for( i = strlen(p->fi_name); i++ < 12 ; *s++ = ' ' )
                        ; /* Pad out the name field */

                sprintf( s, " %6ld ", p->fi_fsize );

                EOS(s) ;
                sprintf(s, "%2d-%02d-%02d %2d:%02d:%02d",
                                C_MONTH(p), C_DAY(p), C_YEAR(p)-1900,
                                C_HR(p),    C_MIN(p), C_SEC(p)    );
                s   += 17 ;
                *s++ = ' ';

                if( IS_READONLY(p) ) *s++ = 'R' ;
                if( IS_HIDDEN(p)   ) *s++ = 'H' ;
                if( IS_SYSTEM(p)   ) *s++ = 'S' ;
                if( IS_LABEL(p)    ) *s++ = 'L' ;
                if( IS_SUBDIR(p)   ) *s++ = 'D' ;
                if( IS_DIRTY(p)    ) *s++ = 'M' ;
                *s = 0;
        }
        return( s - startstr );
}

/*----------------------------------------------------------------------*/

haswild(s)
char    *s;
{
        /*      Return true if s has a '*' or '?' in it   */

        for( ; *s ; s++)
            if( *s == '*' || *s == '?' )
                    return 1;
```

(Continued on next page)

Listing One

```
            return 0;
}

/*-------------------------------------------------------------------------*/

ecmp( s1p, s2p )
char    **s1p, **s2p;
{
            /*  Comparison routine for sorting a directory by extension  */

            char     *s1, *s2;
            int      rval;

            for( s1 = *s1p; *s1 && *s1 != '.' && *s1 != ' '; s1++ )
                    ;
            for( s2 = *s2p; *s2 && *s2 != '.' && *s2 != ' '; s2++ )
                    ;

            if( rval = strcmp(s1, s2) )        /* If the extensions don't match */
                    return rval ;              /* return the strcmp value       */

            for(s1=*s1p, s2=*s2p ; *s1==*s2 && *s1 && *s1 != '.'; s1++, s2++)
                    ;

            return(  (*s1 == '.' ? 0 : *s1)  -  (*s2 == '.' ? 0 : *s2) );
}

/*-------------------------------------------------------------------------*/

printdir(dirc, dirv)
int       dirc    ;
char      **dirv ;
{
          if( !Unsorted )
                    qsort( dirv, dirc, sizeof(*dirv), Ext_sort ? &ecmp : 0);
          ptext( dirc, dirv, stdout, Num_cols, 80/Num_cols,
                                  (dirc/Num_cols) + (dirc % Num_cols != 0) );
}

/*-------------------------------------------------------------------------*/

char     *fixup_name( name, regs, info )
char               *name;
REGS               *regs;
FILE_INFO          *info;
{
         /*  If the name specifies an implicit file (ie. it asks for
          *  the directory rather than the files in the directory),
          *  modify it to ask for files (eg. ".." becomes "..\*.*").
          */

         static char          buf[80], *p ;

         if( !find_first( name, ALL, regs) )
         {
                 /* If the requested name was found, and that name is
                  * a sub directory, append  \*.* to the requested name
                  */

                 if( !IS_SUBDIR(info) )
                         return( name );

                 strncpy( buf, name,  80 );
                 strncat( buf, "\\*.*",80 );
                 return ( buf );
         }
         else
```

```
	{
		/* If you didn't find anything, and a non-explicit
		 * directory is requested (ie. "..\.." or "b:" ) then
		 * set up the name to get the root directory on the
		 * indicated disk.
		 */

		if( name[1] == ':' )		/* Take care of the      */
		{				/* disk designator first */
			buf[0] = name[0];
			buf[1] = ':' ;

			if( !*(name += 2) )
			{
				buf[2] = '*';	/* If the current dir   */
				buf[3] = '.';	/* on another disk is   */
				buf[4] = '*';	/* requested...         */
				buf[5] =  0 ;
				return( buf );
			}
			else
				buf[2] =  0 ;
		}

		for( p = name ; *p ; p++ )
			if( !(*p == '.' || *p == '\\' || *p == '/'))
				return( name );

		strncat( buf, "\\*.*", 80 );
		return( buf );
	}
}

/*----------------------------------------------------------------------*/

add_entry( p )
FILE_INFO		*p;
{
	/*	Add an entry to the directory array (dirv) and update
	 *	various associated globals. Hidden files are not
	 *	printed unless -a was given on the command line.
	 *	(which will cause List_all to be set). As with
	 *	Unix, file names starting with the character '.' are
	 *	also not printed unless -a is specified. If -d was
	 *	given on the command line (Dirs_only will be true)
	 *	and files won't be printed.
	 */

	char		buf[64];
	register int	strlen;

	if( !List_all && (IS_HIDDEN(p) || *p->fi_name == '.')  )
		return 1;

	if( IS_SUBDIR(p) )			/* Entry is a directory      */
		Numdirs++;			/* Adjust directory count    */

	else if( Dirs_only )			/* Entry is a file           */
		return 1;			/* Don't print files         */

	else if( !IS_LABEL(p) )			/* Entry is a file           */
		Numfiles++;			/* Adjust file count         */


	strlen = dirtoa( buf, p, Longfmt );
	if( ++Dirc > MAXDIR || !(*Dirv = malloc( strlen + 1 )) )
		return 0;

	strcpy( *Dirv++, buf );
	Total += p->fi_fsize;
	return 1;
}
```

(Continued on next page)

The word is out. dBASE III is the world

"dBASE III is the most mature, best performing business programming language available on any computer."

Adam Green
Personal Software
August, 1984

"With its impressive specifications, dBASE III is clearly designed for the future. With 128 fields per record, a user could feasibly have access to 1,280 fields of information, opening the micro to applications heretofore reserved for large systems."

Edward Bride
Software News
July, 1984

"Ashton-Tate has created dBASE III, the connoisseur's database management system."

Russell Lipton
Business Computer Systems
October, 1984

"Enter dBASE III. Not only is this whiz of a data base loaded with features customers asked for, it sports an enhanced language that brings database programming into a new dimension."

Daryl Rubin
Computer Language
January, 1985

"dBASE III makes that wish list a reality..."

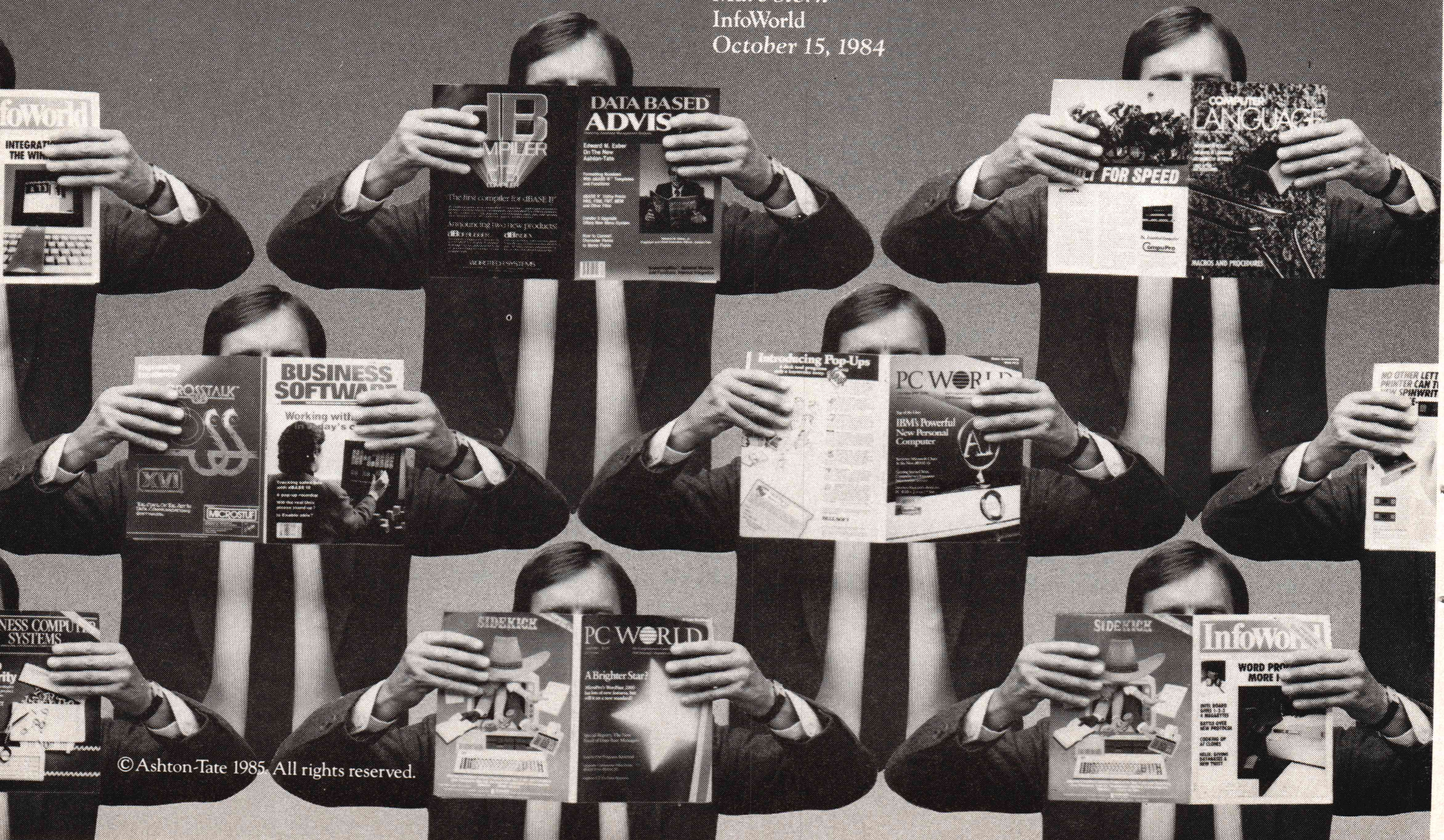
Marc Stern
InfoWorld
October 15, 1984

"Ashton-Tate has done it again. dBASE III should assure the company an extended stay at the top of the microcomputer database management system market."

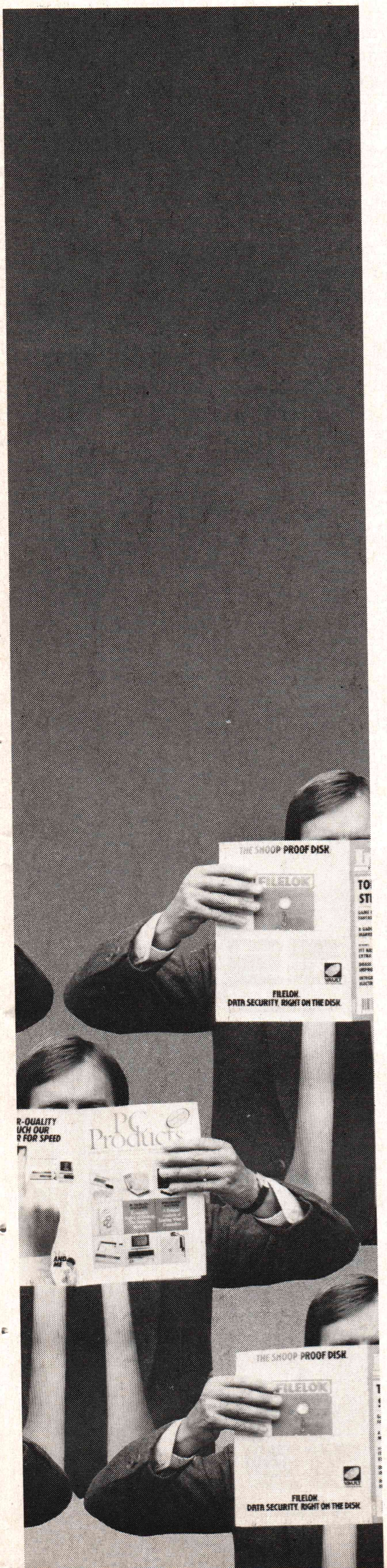
Rowland Archer
Business Computer Systems
November, 1984

"It's a classy product. It's easy to use, has incredible power, polish and finesse. It's an easy program for the novice. There is untapped power for the experienced user."

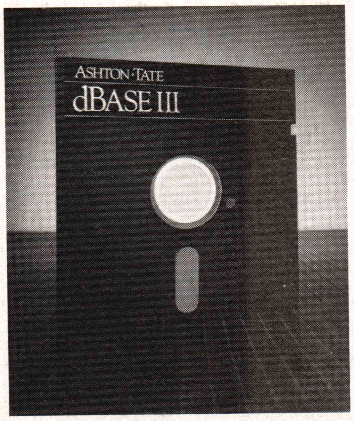
Robert Byers
Data Based Advisor
September, 1984



's best data management software.



dBASE III™ is the industry standard for managing information on personal computers. Written to take full advantage of the capabilities of the IBM® PC, Personal Computer AT,™ compatible computers and other popular personal computers, dBASE III offers virtually limitless data-handling capabilities. And, for novices, a built-in Assistant feature makes dBASE III the data management system of choice.



With so many great reviews, dBASE III is the clear choice in data management.

For a dealer near you call (800) 437-4329, ext. 2333. In Colorado call (303) 799-4900, ext. 2333.

dBASE III and Ashton-Tate are trademarks of Ashton-Tate. IBM is a registered trademark and Personal Computer AT is a trademark of International Business Machines Corp.

dBASE III. The data management standard.

ASHTON·TATE
S O F T W A R E

Circle **no. 5** on reader service card.

Listing One

```
/*-------------------------------------------------------------------*/
main(argc, argv)
int        argc;
char       **argv;
{
           /* Get a directory. Given a list of files on the command line
            * (file names may be ambiguous) will print a sorted directory
            * of those files along with a total file and byte count.
            */

           static REGS        regs           ;     /* Needed for DOs calls */
           static FILE_INFO   info           ;     /* DOs puts dirs here   */
           static char        *name  = "*.*";      /* File name            */

           gregs  ( &regs  );                  /* Initialize the regs structures */
           regs.x.dx = (word) &info;           /* Change the Disk Transfer Addr  */
           doscall( SETDTA, &regs );           /* to point at info structure     */


           argc = getargs( argc, argv, Argtab, TSIZE );

           if( Longfmt )                   /* -1 is always printed in 1 column */
                   Num_cols = 1;           /* even if -cN was given on the cmd */
                                           /* line                             */

           if( argc >= 2 )                 /* Put the requested file name into */
           {                               /* "name." The default is *.*       */
                   name = *(++argv);
                   if( argc == 2  &&  !haswild(name)  )
                           name = fixup_name( name, &regs, &info );
           }

           do
           {
                   if( !find_first(name, Files_only ? ALL_FILEs : ALL, &regs))
                   {
                           if( !add_entry(&info) )
                                   break;

                           if( haswild(name) )
                                   while( !find_next( &regs ) )
                                           if( !add_entry(&info) )
                                                   goto abort;
                   }
                   name = *(++argv);

           } while( --argc > 1 );

abort:     printf   ("\n");
           printdir ( Dirc, Dirs );
           printf   ("\n");
           if( Numfiles )
           {
                   printf("%d file%s (%ld bytes, %d K)",
                                           Numfiles, Numfiles == 1 ? "" : "s",
                                           Total, Total/1024 );
           }

           if( Numdirs  )
           {
                   if( Numfiles )
                           printf(", ");

                   printf("%d director%s",Numdirs, Numdirs ==1 ? "y" :"ies");
           }
}
```

End Listing One

Listing Two

```
/*----------------------------------------------------------------------+
 *                                                                      |
 *        MYDOS.H        Various defines for talking to dos             |
 *                                                                      |
 *----------------------------------------------------------------------+
 *
 *    Typdefs for using dos(). Note that this structure can be used
 *    by the various routines supplied by Lattice (intdos, bdos etc.)
 *    but dos() can't use the structure defined in Lattice's dos.h.
 *
 *    Since "REGS" and "byte" are both also defined in dos.h, you
 *    shouldn't #include both of them in the same place.
 */

typedef short   word;
typedef char    byte;

struct LREG { word  ax,    bx,    cx,    dx,     si, di, es, cs, ss, ds;};
struct SREG { byte  al,ah, bl,bh, cl,ch, dl,dh;                         };

typedef union
{
        struct LREG     x ;
        struct SREG     h ;
}
REGS;

/*----------------------------------------------------------------------
 *      Stuff needed to get a directory from MSDOS
 *
 *  The FILE_INFO type structure is filled by a find first or find
 *  next command (dos system calls 0x4e and 0x4f).
 */

typedef struct
{
        char    fi_resv[21];     /* Bytes 0-20   Reserved by DOS         */
        char    fi_attrib;       /* Byte  21     File attribute          */
        short   fi_time;         /* Bytes 22-23  Create/update time      */
        short   fi_date;         /* Bytes 24-25  Create/update date      */
        long    fi_fsize;        /* Bytes 26-27  File size in bytes      */
        char    fi_name[13];     /* Bytes 28-40  File name & extension */
}
FILE_INFO;


/*
 *   Macros to extract information from a FILE_INFO. In all these macros
 *   the argument "p" is a poshorter to a FILE_INFO structure. Note that
 *   the C_YEAR and C_SEC macros compensate for MSDOS wierdnessess.
 *
 * IS_READONLY(p)       File is read only.
 * IS_HIDDEN(p)         File is invisible in normal directory searches
 * IS_SYSTEM(p)         File is a system file
 * IS_LABEL(p)          Info is a volume label, not a file.
 * IS_SUBDIR(p)         File is a directory
 * IS_DIRTY(p)          True when file is written to and closed, set to
 *                      False by the program backup.
 * C_HR(p)              Hour of last update or create   (0-23)
 * C_MIN(p)             Minute of last update or create (0-59)
 * C_SEC(p)             Second of last update or create (0-59)
 * C_YEAR(p)            Year of last update or create   (1980-2099)
 * C_MONTH(p)           Month of last update or create  (1-12)
 * C_DAY(p)             Day of last update or create    (1-31)
 */

#define READONLY        0x01               /* Attribute bits        */
#define HIDDEN          0x02
#define SYSTEM          0x04
```

(Continued on next page)

Listing Two

```
#define LABEL           0x08
#define SUBDIR          0x10
#define DIRTY           0x20

#define ALL          (READONLY | DIRTY | SYSTEM | HIDDEN | SUBDIR | LABEL)
#define ALL_FILES    (READONLY | DIRTY | SYSTEM | HIDDEN )
#define NORM_FILES   (READONLY | DIRTY )

#define IS_READONLY(p)  ((p)->fi_attrib & READONLY      )
#define IS_HIDDEN(p)    ((p)->fi_attrib & HIDDEN        )
#define IS_SYSTEM(p)    ((p)->fi_attrib & SYSTEM        )
#define IS_LABEL(p)     ((p)->fi_attrib & LABEL         )
#define IS_SUBDIR(p)    ((p)->fi_attrib & SUBDIR        )
#define IS_DIRTY(p)     ((p)->fi_attrib & DIRTY         )

#define C_HR(p)         ( ((p)->fi_time >> 11) & 0x1f )
#define C_MIN(p)        ( ((p)->fi_time >>  5) & 0x3f )
#define C_SEC(p)        ( ((p)->fi_time <<  1) & 0x3e )
#define C_YEAR(p)       ((((p)->fi_date >>  9) & 0x7f ) + 1980)
#define C_MONTH(p)      ( ((p)->fi_date >>  5) & 0x0f )
#define C_DAY(p)        ( ((p)->fi_date       ) & 0x1f )

/*----------------------------------------------------------------------
 *      Directory related BDOS function numbers
 */

#define FINDFIRST       0x4e
#define FINDNEXT        0x4f
#define SETDTA          0x1a
#define GETDTA          0x2f
```

End Listing Two

Listing Three

```
        TITLE   DOS INTERFACE FUNCTION
        SUBTTL  Copyright 1985 by Allen I. Holub
        NAME    DOS
        INCLUDE DOS.MAC

;----------------------------------------------------------------------
;
; Data types: The REGS structure is defined as type allregs in mydos.h
;

REGS    STRUC                   ; Structure used to transfer register
AX_REG  DW      0               ; contents. Note that this structure is
BX_REG  DW      0               ; a supperset of that defined in the
CX_REG  DW      0               ; dos.h supplied by Lattice. This particular
DX_REG  DW      0               ; structure is defined in \include\mydos.h
SI_REG  DW      0
DI_REG  DW      0
ES_REG  DW      0
CS_REG  DW      0
SS_REG  DW      0
DS_REG  DW      0
REGS    ENDS

OFF     EQU     4               ; Offset to arguments

;----------------------------------------------------------------------
;
; name:         dos  --  Do a dos function call
;
; synopsis:     #include <mydos.h>
;
```

(Continued on page 36)

 (Listing Continued, text begins on page 20)

Listing Three

```
;                 status = dos( regsp );
;                 int      status   ;      /* returned status register    */
;                 REGS     *regp    ;      /* pointer to register struct */
;
; description:    This function is a more useful version of the bdos()
;                 function provided with the lattice C compiler. It
;                 saves the existing machine state, replaces the
;                 contents of all registers except CS & SS from the
;                 structure pointed to by regp, does an int 21, loads
;                 the registers back into the structure, restores the
;                 machine state and returns.
;
; notes:          This function will only work with the SMALL model, since
;                 it assumes a 16 bit pointer. The stack frame is
;                 non-standard too (ie the BP holds the structure pointer
;                 arg rather than the frame pointer).
;
          PSEG

          PUBLIC  DOS
DOS       PROC    NEAR

          PUSH    BP                ; Save the old stack frame.
          MOV     BP,SP             ;
          MOV     BP,[BP+OFF]       ; BP = pointer to register structure.

          PUSH    BX                ; Save the current machine state. No
          PUSH    CX                ; point in saving AX because it's
          PUSH    DX                ; going to be used for the return
          PUSH    SI                ; value.
          PUSH    DI
          PUSH    ES
          PUSH    DS

          MOV     AX,SS:[BP].AX_REG         ; Set up a new machine state
          MOV     BX,SS:[BP].BX_REG         ; using the contents of the
          MOV     CX,SS:[BP].CX_REG         ; REGS structure. Don't modify
          MOV     DX,SS:[BP].DX_REG         ; the SS or CS registers.
          MOV     SI,SS:[BP].SI_REG
          MOV     DI,SS:[BP].DI_REG
          MOV     ES,SS:[BP].ES_REG
          MOV     DS,SS:[BP].DS_REG

          PUSH    BP                        ; Make the DOS call. Save the BP
          INT     21H                       ; out of irrational paranoia.
          POP     BP

          MOV     SS:[BP].AX_REG,AX         ; Now update the original structure
          MOV     SS:[BP].BX_REG,BX         ; to reflect the return values from
          MOV     SS:[BP].CX_REG,CX         ; the dos call.
          MOV     SS:[BP].DX_REG,DX
          MOV     SS:[BP].SI_REG,SI
          MOV     SS:[BP].DI_REG,DI
          MOV     SS:[BP].ES_REG,ES
          MOV     SS:[BP].CS_REG,CS
          MOV     SS:[BP].DS_REG,DS
          MOV     SS:[BP].SS_REG,SS

          POP     DS
          POP     ES
          POP     DI                ; Restore the previous machine state
          POP     SI
          POP     DX
          POP     CX
          POP     BX
```

```
        LAHF                    ; Return the flags:
        MOV     AL,AH           ;       Move flags to LSB
        XOR     AH,AH           ;       Clear high bytes

        POP     BP              ; Restore the previous stack frame's
        RET                     ; frame pointer and return.
DOS     ENDP

;------------------------------------------------------------------
;
; name:         gregs   --  Initialize a REGS structure to the current
;                           register contents.
;
; synopsis:     #include <mydos.h>
;
;               gregs( regp )
;               REGS    *regp   ;       /* pointer to register struct */
;
; description:  This function is used before before calling dos().
;
; notes:        This function will only work with the SMALL model.
;

        PUBLIC  GREGS
GREGS   PROC    NEAR

        PUSH    BP                  ; Save the old stack frame.
        MOV     BP,SP               ;
        MOV     BP,[BP+OFF]         ; BP = pointer to register structure.

        MOV     SS:[BP].AX_REG,AX   ; Initialize the structure pointed
        MOV     SS:[BP].BX_REG,BX   ; to by bp.
        MOV     SS:[BP].CX_REG,CX
        MOV     SS:[BP].DX_REG,DX
        MOV     SS:[BP].SI_REG,SI
        MOV     SS:[BP].DI_REG,DI
        MOV     SS:[BP].ES_REG,ES
        MOV     SS:[BP].CS_REG,CS
        MOV     SS:[BP].DS_REG,DS
        MOV     SS:[BP].SS_REG,SS

        POP     BP                    ; Restore the previous stack
        RET                           ; frame and return.

GREGS   ENDP
        ENDPS
        END
```

End Listing Three

Listing Four

```
#include <stdio.h>

/*
 *                PTEXT.C: Multi-column print utility
 *
 *      Copyright (c) 1985 Allen I. Holub,  all rights reserved.
 *    This program may be reproduced for personal, non-profit use only
 *
 *     Bugs and Features:
 *
 *     1) Pr_line assumes that the output device supports backspace
 *        ('\b' == ^H ).   This is only a problem when  the  source
 *        has underlined text implemented as:
 *                   texttexttext\r  ________
 *        We can't just ouput a '\r' in this case because we may
 *        not be in the leftmost column. A \r should get us to the
 *        left edge of the current column.
 *
```

(Continued on next page)

Listing Four

```
 *      2) When printing multi-column stuff. If a line is truncated it
 *         will run into the column to its right. That is, there is no
 *         seperator between columns other than the  whitespace needed
 *         to pad the column out to a particular width. If no  padding
 *         is necessary (ie. the lines have been truncated), then  the
 *         columns will run together.
 *
 *      3) When an ESC is found, the escape and the next three characters
 *         take up no space in the output. This lets us print out the
 *         various SGR commands without messing up the column width.
 *
 *      4) Strange things happen to an IBM screen when the leftmost
 *         character on a line is printed with underline. To compensate
 *         for this, a single blank is printed as the left-most
 *         character on every line if IBM is #defined.
 */

#define ESC      0x1b
#define IBM      1

/*-----------------------------------------------------------------------*/

ptext(linec, linev, outfile, numcols, colwidth, numrows)
int      linec, numcols, colwidth;
char     **linev;
FILE     *outfile;
int      numrows;
{
         /*      Print out the array of strings "linev" which consists of
          *      linec entrys. Output is sent to "outfile" formated as
          *      follows:
          *
          *      "numcols"   : number of columns
          *      "colwidth"  : width of a column in characters. Any text
          *                       longer than colwidth is truncated off.
          *      "numrows"   : columns are numrows long. The left-most
          *                       column is printed in its entirety
          *                       first then the next column in its
          *                       entirety, and so on.
          */

         register int     j ;
         register char    **lineend, **line, **nextline ;

         lineend = &linev[linec-1];        /* Last linev entry to print    */

         for( j = numrows ; --j >= 0 ; )
         {
#ifdef IBM
                 putc( ' ', outfile );
#endif
                 for( line = linev++;  line <= lineend; line = nextline)
                 {
                         nextline = line + numrows;

                         /*   Print the line. Don't pad the rightmost    */
                         /*   column.                                    */

                         pr_line( *line, outfile, colwidth,
                                                  nextline <= lineend);
                 }

                 fputs("\n" , outfile);
         }
}
```

(Continued on page 40)

BRIEF™

"BRIEF Is The Best Editor I Have Ever Used. I Switched. I Use BRIEF Regularly."

Steve McMahon*

Choosing the right program editor is an extremely important decision. After all, more time is spent with your program editor than with any other category of software. A good program editor helps to stimulate creativity and produce programs of superior design. A poor program editor, on the other hand, can be worse than none at all. It can slow you down, get in your way, and make even the simplest tasks difficult.

BRIEF - PROGRAM EDITING *YOUR WAY*

Every programmer has an individual style that makes their work unique. Most program editors compromise that style by forcing the programmer to conform to their methods. Not so with BRIEF.

BRIEF's most powerful feature is its ability to conform to your way of programming. BRIEF can be easily tailored/customized to your individual preferences. For example, keyboard reassignment allows the keyboard to be used in whatever way you prefer. Keystroke macros allow long sequences of editor directives to be repeated by pressing a single key. Plus you can simultaneously work with numerous program and data files, and configure many windows to control BRIEF's visual presentation.

75% OF THE EXPERTS WHO HAVE TRIED BRIEF SWITCHED!

(Call For Details)

"A BONA FIDE UNDO" Steve McMahon - BYTE REVIEW "BUILD YOUR DREAM EDITOR" MARCH 1985

Here are Steve McMahon's exact words: "...BRIEF implements a true undo facility, by default allowing command-by-command recovery from the last 30 commands...The number of commands you want to be able to undo can be changed." (up to 300) "Only with BRIEF, though, was it possible to undo a macro that produced 4000 words of text with a single keystroke."

EVERY FEATURE YOU WANT

BRIEF supports practically every feature you've ever seen, and some you probably haven't thought of yet.

- Edit Multiple Large Files
- Windows (Tiled & Pop-up)
- Unlimited File Size
- Full UNDO (N Times)
- True Automatic Indent for C
- Exit to DOS Inside BRIEF
- Compile programs inside BRIEF
- Uses All Available Memory
- Intuitive Commands
- Tutorial
- Repeat Keystroke Sequences
- 15 Minute Learning
- Reconfigurable Keyboard
- Online Help
- Search for "Regular Expressions"
- Mnemonic Key Assignments
- Horizontal Scrolling
- Comprehensive Error Recovery
- And ... a Complete, Compiled, Programmable and Readable Macro Language.

```
================================ make.c ================================
int        handle = 0;

main (argc, argv

int             argc;
============= fsa.h ================= makefile.h =======================
#include "..\include\ctype.h"        | /**
                                     |  **             makefile.h:
typedef  struct                      |  **
{                                    |  **        This is the definitions fil
   short    action,                  |  **   Hopefully, it won't be unreasonab
            state;                   |  **   that have been written.
} Fsa;                               |  **/
#ifdef   FSA_MAIN                    |
Fsa      fsa[][8] = {/* Alphanum   Co| typedef struct cmd_struct
                                     | {
/* State 0. */          0, 2,      10|    char        *cmd_text;
/* State 1. */          10, 0,     10|
/* State 2. */          0, 2,      1,|    struct cmd_struct *next_cmd;
/* State 3. */          0, 5,      11| } *Cmd_Ptr, Cmd;
/* State 4. */          0, 4,      0,|
Mismatched open parenthesis.              Line: 11   Col: 17      2:17 pm
```

A TYPICAL BRIEF SCREEN

Notice there are three windows on the screen simultaneously and each one is viewing a different file. The mainline of a C program is visible in the uppermost window; the programmer has run a syntax checking macro which found a mismatched open parenthesis in the arguments to the mainline. The other two windows show header files containing information crucial to the design of the program. BRIEF can have an unlimited number of windows and files accessed simultaneously.

BRIEF'S ONLY LIMITATIONS ARE THE ONES *YOU* SET

BRIEF's capabilities are limited only by your imagination. Once you've used BRIEF, you will NEVER want another editor.

TRY BRIEF RISK FREE FOR 30 DAYS!

Order BRIEF ($195) and try it for 30 days with a full money-back guarantee. CALL 800-821-2492

AVAILABILITY: IBM-PC and Compatibles, AT, & TANDY 2000

Solution Systems™

335-D Washington St., Norwell, MA 02061 (617) 659-1571

*Steve McMahon's quote courtesy Suntype Publishing Systems. BYTE review by Mr. McMahon may be found in BYTE Magazine March 1985.

Circle no. 102 on reader service card.

 (Listing Continued, text begins on page 20)

Listing Four

```
/*-------------------------------------------------------------------------*/

static pr_line( str, stream, width, padded )
register char   *str;
FILE            *stream;
int             width, padded;
{
        /*      Print out "str" into "stream" padding it to "width"
         * columns wide. Non-printing characters and 3 printing characters
         * immediatly following an ESC are not counted as having printed.
         * '\n' characters are treated as line teminators but are not
         * printed. If "padded" is 0 then no padding is done, though the
         * line will still be truncated if it's too long.
         */

        int     col = 0 ;

        while( col < width  &&  *str )
        {
                if ( *str == '\n' )
                        break;

                else if( *str == '\r' )
                {
                        /* Back up to the left edge of the current column
                         */

                        while( col >  0 )
                        {
                                --col;
                                putc('\b' , stream);
                        }

                        str++;
                }
                else if( *str == '\t' )         /* expand tabs */
                {
                        str++ ;
                        col++ ;
                        putc( ' ' , stream );

                        while( (col % 8)  &&  col < width)
                        {
                                putc( ' ' , stream );
                                col++;
                        }
                }
                else
                {
                        if( *str == ESC )
                        {
                                putc( *str++ , stream );
                                if( !*str )
                                        break;

                                putc( *str++ , stream );
                                if( !*str )
                                        break;

                                putc( *str++ , stream );
                                if( !*str )
                                        break;
                        }

                        else if (*str == '\b')
                                --col ;
```

```
                        else if( *str >= ' ' )
                                ++col ;

                        putc( *str++ , stream );
                }
        }

        if (padded)
                while( col++  <  width )
                        putc( ' '  , stream );
}
```

End Listing Four

Listing Five

```
/* Example of how to get and restore the MSDOS Disk Transfer address
 * using the dos() and gregs() functions defined in listing 3
 */

#include <mydos.h>

#define SETDTA  0x1a
#define GETDTA  0x2f

main()
{
        static FILE_INFO info;
        static REGS      oregs, nregs;

        gregs( &oregs );                /* Get the old DTA. It will be put */
        oregs.h.ah = GETDTA;            /* into ES:BX of oregs.            */
        dos( &oregs );

        gregs( &nregs );
        nregs.x.dx = (word) &info;      /* Change the Disk Transfer Addr  */
        nregs.h.ah = SETDTA ;           /* to point at info structure     */
        dos( &nregs );

        /*   . . .   */

        oregs.x.ds = oregs.x.es;        /* Now put the DTA back. You have  */
        oregs.x.dx = oregs.x.bx;        /* to copy ES:BX into DS:DX first. */
        nregs.h.ah = SETDTA ;
        dos( &oregs );
}
```

End Listings

Build a Custom PC or Clone

by Jim Kronman

I have owned an 8-bit S-100 microcomputer since 1978. Old habits are hard to kick, so I set about assembling a PC-compatible computer the same way I built my S-100 system. I individually selected each component of the system to suit my specific needs. I now have a PC XT work-alike that is totally "IBM compatible," costs substantially less than the closest off-the-shelf equivalent, and fits my requirements exactly.

My approach may not be for everyone. I am pleased with my completed project, but you should consider the pros and cons before taking the plunge yourself. Before going into the advantages of rolling your own PC, I'll point out some of the drawbacks you should consider.

Careful shopping, gentle persuasion with vise grips and a lot of patience yield an XT work-alike at a PC price.

You will need patience. If you must have a computer tomorrow, go buy something off the shelf of your local computer store. I ordered pieces from five separate mailorder suppliers (including IBM), and it was over five weeks from the time I placed the orders by telephone until I first booted the system. At that point, I was still missing a vital element, but more on that subject later.

Do not attempt to assemble your own PC if you are not experienced with microcomputer hardware or are timid when handling equipment costing, in the aggregate, up to several thousand dollars. When you put Manufacturer A's board and Manufacturer B's board into your computer, you alone are responsible for determining that the boards are compatible both with each other and with the PC itself. The vendors and manufacturers usually will try to help if you have a problem, but they may never have encountered exactly the configuration you have assembled and therefore may be unable to determine the cause of your problem.

Now that you have received fair warning of some of the potential pitfalls, I'll shamelessly promote the advantages of building your own PC. You can save a bundle of money when you compare your homegrown computer to one with the same functionality in a store: for the same amount of money, you can get a lot more capability. With my project now complete, I realize, too, that there are other benefits I had not anticipated when I began.

An obvious reason to undertake this adventure is to assemble a system exactly suited to your needs. One of my prime requirements was to have a Selectric style keyboard, which IBM does not offer with the PC. I did not want to buy the standard IBM PC because I would end up throwing away a perfectly good (if poorly laid out) key-

Jim Kronman, 6085 Venice Blvd., No. 16, Los Angeles, CA 90034 (213) 558-3281

board and buying a replacement. (As far as I know, IBM will not sell the PC System Unit without a keyboard.)

I purchased the Keytronic KB5151 deluxe keyboard with a Selectric style layout and separate numeric and arrow keypads; it also has the function keys across the top of the keyboard so they will align with the function key legends in row 25 of the display in many application programs. My one criticism of the Keytronic design was that the "touch" was too soft; sometimes the slightest tap would cause a character to appear on the screen. This problem was easy to remedy. I carefully took each keytop off of the keyboard, stretched the spring under the keytop to about 0.8 inches, and then replaced the keytop. Do one key at a time and you should have no problems with this slight modification; it's well worth the time!

I had decided that I wanted the capability of a hard disk and that I wanted it internal to the computer. I also did not like the crowded arrangement in the eight-slot PC XT or the XT clones. These considerations and others led me to opt for a custom assembly with a five-slot PC motherboard. I'm sure you can think of plenty of your own reasons to customize, too.

While my project was in progress, I had occasion to help a friend with an IBM PC that had a failed power supply. What I have come to call the "$290 Fuse Syndrome" is a strong argument to buy as few components as possible from IBM. Big Blue's warranty policy (which I have seen reviewers praise in several magazine articles) is to replace defective components on the spot and without question. That's great while you are within the warranty period and the tab is on Blue, but unfortunately the policy continues and is *mandatory* after the warranty expires and you are paying the bill. If you try to obtain a schematic or other service information for an IBM PC component that is designated a FRU (Field Replaceable Unit), you will encounter as firm a stonewall from IBM as you will ever want to see. Even if you figure out who made the component for IBM, you can't get any help because IBM has its vendors contractually bound to stony silence as well.

The $290 Fuse Syndrome is so named because inside of each IBM power supply is a standard slow-blow radio-type fuse—in a fuse holder! If this fuse blows (or even simply fails), IBM's policy is that the supply is terminally (if you'll pardon the expression) ill and must be replaced. A PC XT (130 watts) power supply costs $290 when ordered from IBM; as I write this article, the going rate for a clone replacement seems to be around $170. By purchasing your power supply, plug-in boards, disk drives, and so on from other sources, you can obtain servicing information in most cases. If the availability of servicing information is vital to you, check with each manufacturer before you buy.

My original plan was to purchase a PC motherboard, the case, and a 130 watt power supply from IBM. This would have given the machine the appearance of being "true Blue." IBM was one step ahead of me: I was unable to obtain the entire case from

IBM, even though all the pieces are listed in the IBM PC Service Manual's illustrated parts breakdown and the Greencastle Parts Depot accepted my order. I did buy the power supply, which has given me no problems to date (although the first supply sent to me was dead on arrival), but I regret not saving the $100 by buying a supply from a third party.

My inability to obtain the case from IBM was frustrating; after the IBM Parts Center accepted my order for all of the pieces, it simply failed to deliver all of them. I never received notification that the entire order would not be filled. As the weeks dragged by, I assembled all of the components of the by now functional computer on a 24 × 27-inch piece of particle board. When I finally found out what was happening with my order from IBM, the Parts Depot at least let me return for credit the pieces of the case I had obtained and could not use.

I found a clone chassis made in Taiwan for about half the price of the IBM parts, and after a small amount of persuasion with vise grip pliers to straighten out a bracket, everything fit fine. The case is identical in appearance to the IBM from the front, except for the absence of the famous logo, and the back panel has a variety of cutouts adaptable to various hardware configurations.

One piece of the system that had to be genuine IBM was the motherboard, which contains the IBM ROM BIOS, thus assuring total PC compatibility. Although some of the clone boards might be close to 100% compatible, the difference in price was not worth the risk. If you have reliable information about a specific non-IBM motherboard that meets your needs, then go for it!

One other specific piece of hardware made the choice of the PC (rather than XT) motherboard attractive: Maynard Electronics' "Sandstar Series" WS-2 disk controller board supports up to four floppy disk drives and two 10 Mb hard disks and takes up only a single slot. An added advantage of this configuration is that the Maynard hard disk software driver is in a PROM that is installed on the main circuit board, occupying address space in the F0000h segment, while the IBM XT controller is a separate board and has its driver in the middle of the C0000h segment. Thus, by not using the XT controller or any other board with ROM code in the C0000h segment, you make available 64K of memory space along with the D0000h and E0000h segments, which allows you to install a 192K RAM disk without using any of the 640K available to DOS. Use caution in dealing with the C0000h through E0000h segments; according to the IBM Technical Reference Manual (page 2-11), plug-in boards can use the area from C2000h to F4000h for software drivers that the ROM BIOS will recognize at power-on.

When you do not buy the System Unit from IBM, you do not receive the IBM PC Guide to Operations manual, the BASIC manual, or the IBM Diagnostic Disk. You may purchase these items from IBM if you need them. If you purchase the IBM Technical Reference Manual, you will not need the Guide to Operations. (If you need this Guide to Operations, you probably shouldn't be assembling your own computer!) I highly recommend that you purchase the Technical Reference Manual; it is the definitive source of information on the ROM BIOS function calls, containing the source code for the entire ROM BIOS.

After I completed my project (and after my friend purchased a replacement power supply), I discovered that Howard W. Sams publishes a "Computerfact" package for the IBM PC. It costs $39.95 and includes schematics, circuit board photos with component identifications and trouble-shooting instructions. The package covers the PC motherboard, power supply, keyboard, display adapters, floppy disk controller, and a few other optional cards.

You must also purchase an operating system separately. The various choices are PCDOS 2.1 or 3.0 from IBM and Digital Research's CP/M-86, Concurrent CP/M v3.1, or Concurrent DOS v3.2. By far the most software is available for PCDOS. Because v3.0 seems to offer the single user little advantage over v2.1, I selected PCDOS 2.1. The choice is yours. You can use more than one operating system (but not simultaneously).

That is about all there is to tell about my experiences. Not only would I do it again this way if I had to (less the mistakes, of course), I would not consider doing it any other way! What follows is a summary of my advice and some useful tabulations of information, components, and sources.

If you intend to assemble your own PC:

(1) Be confident of your skills and knowledge in microcomputers.
(2) Be prepared to wait; this isn't a project for the impatient.
(3) Select your components carefully; make sure each item is compatible with the others.
(4) Understand each vendor's return policy before you order, just in case.
(5) Select your software with as much or more care as your hardware.
(6) If you have a specific software application in mind, make sure the hardware you select is compatible.

The components you will need (as a minimum) are:

(1) Motherboard, with CPU, ROM BIOS, memory, keyboard interface, and support circuits
(2) Power supply, 62 to 130 watts (130 watts recommended)
(3) Display adapter card
(4) Monitor compatible with display adapter
(5) Disk controller card (possibly a second card for hard disk)
(6) Case
(7) Keyboard
(8) Floppy disk drive, 5¼-inch DSDD, 48 tpi, PC-compatible

Optional components include (but are not limited to):

(1) Multifunction card (additional memory, battery run clock, serial and parallel ports, game port, etc.)
(2) MODEM card
(3) Math coprocessor chip (8087)
(4) Coprocessor board (Z80 CP/M-80, Apple, 10286, etc.)

The following items are available from IBM:

(1) PC motherboard with 64K memory installed, P/N 8654213, $640.00
(2) PC XT motherboard with 128K memory installed, P/N 8529254, $750.00
(3) Speaker with mounting bracket, P/N 8529143, $8.15
(4) Guide To Operations manual
(5) Technical Reference Manual, P/N 1502234
(6) BASIC reference manual, P/N 6025010
(7) PCDOS 2.1, P/N 6024120
(8) PC Diagnostic Disk, P/N 1502212

All of the above except for the motherboards and speaker should be available through the IBM Product Centers or other retail outlets (Sears Business Centers are usually a good place to check). The replacement parts (motherboard, etc.) are available from the IBM Parts Depot in Greencastle, IN. It is best to deal by telephone: (317) 658-2022. The address is P.O. Box 505, Greencastle, IN 46135. The hardware prices may change; they are included here for reference only.

For the other pieces that you will need for your system, look through the ads in this magazine, *The Computer Shopper*, and *Byte*. Many firms sell PC clone cases, power supplies, keyboards, and so on. I have even found bargains at surplus companies; a firm in the Los Angeles area recently had surplus Keytronics keyboards manufactured with special legends on the keys for just $35!

Happy hunting and happy cloning. I hope this article encourages fellow computer hobbyists to explore the PC bus in the same spirit as the S-100 bus. I have found the experience to be well worth the effort.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 192.**

The Ultimate Parallel Print Spooler

by Don Rindsberg

Don Rindsberg has come up with a printer spooler design incorporating low-cost 6665 (64K) dynamic RAM chips. The result—a 60K print spooler that you can build for about a hundred dollars.

If you are tired of waiting while your parallel printer does its thing, here is a project that will allow you to print and compute at the same time. It is a print spooler that provides a whopping 60K of buffering. The spooler, designed to stand alone, will interface almost any parallel printer port.

The project incorporates "software refresh" of the dynamic RAM memory—a concept little publicized, to my knowledge, except in Motorola's application literature. By using software refresh of RAM, you can eliminate a number of integrated circuits from the design of the spooler. The refresh cycle ties up the microprocessor for less than 10 percent of its time, a penalty easily tolerated when you are controlling a printer, which operates at a snail-like speed from the microprocessor's point of view. You implement software refresh by carefully writing the software so that the 128 consecutive row addresses are read at least every 2 ms: 128 rows in a dynamic RAM must be refreshed, and the maximum allowable time between refreshes is 2 ms.

Don Rindsberg, 5958 S. Shenandoah Rd., Mobile, AL 36608.

I built this spooler to interface my Apple II computer to an Epson parallel printer. However, it should be easily adaptable to virtually any computer with a parallel printer port and any parallel printer. The 60K of buffering holds at least 12 pages of solid, single-spaced text. In the graphics mode, an entire screen is dumped to the spooler in a couple of seconds.

This design uses software refresh of dynamic ROM to create a 60K spooler for about $100.

The Hardware

The 6809 microprocessor, which is now available for about $12, is ideal for our software-refreshed memory system: its internal clock can provide the time-critical signals to the dynamic memory without the use of additional clocks or delay lines. The 6809 (not the 6809E) has its own on-chip clock generator and requires only a crystal in the 3.5 to 4.0 MHz range; a low-cost TV color-burst crystal is a simple and inexpensive choice.

The 6809 produces two clock signals, E (enable) and Q (quadrature), which are both 50 percent duty cycle square waves at a frequency exactly one-fourth that of the crystal. As shown on the timing diagram (Figure 1, page 47), the Q signal leads the E signal by one-fourth of a cycle—the E signal, by the way, is the same as the phase two signal on the 6800 and 6502 microprocessors. We use the rising edge of Q to trigger RAS (row address strobe) on the dynamic RAM. The rising edge of E causes the switch from row addresses to column

addresses, while the falling edge of Q triggers CAS (column address strobe) to the RAM chips. RAS is exerted every cycle to force a refresh, while CAS is exerted only if the address is not an "F" address ($F000–FFFF).

Figure 2 (page 48) shows that we decode address F in the 74LS20, which produces a low when A12–A15 are all high. The 2716 ROM is at $F800–FFFF, and the PIA (parallel interface adapter) is at $F400–F403; address 11 provides the distinction between the ROM and the PIA. We buffered A11–A15 with a 74LS367 because the 6809 has limited drive capabilities. Figure 3 (page 48) shows how the switching between the row addresses A0–A7 and column addresses A8–A15 was done in a pair of 74LS157 chips. The 33 ohm resistors prevent ringing on the RAM's address lines.

The 6821 PIA in Figure 4 (page 50) was designed for applications such as this, having exactly the number of functions we need. It automatically provides the handshaking signals used by parallel printers. When the A side is used as a parallel input, STR from the host computer sets a flag. When the spooler reads the data, the PIA automatically provides a one-cycle negative-going signal, which is used for ACK. On the B side of the PIA, a similar STR signal automatically is generated on a write to port B, and the printer's ACK signal sets a flag to signal that the printer is ready for another character.

Figure 5 (page 50) shows the completed spooler. A 74LS74, used in conjunction with the pause switch, allows manual shutdown of the printout to change paper and so on. This chip changes state at the falling edge of E when you change the switch position. At this time, the opposite half of the ROM is bank-switched into memory, which contains a holding routine. The two halves of the ROM are selected by its A10 line. The power supply (Figure 6, page 51) provides regulated 5 volts at somewhat less than 500 ma.

The Software

The software has two unusual requirements:

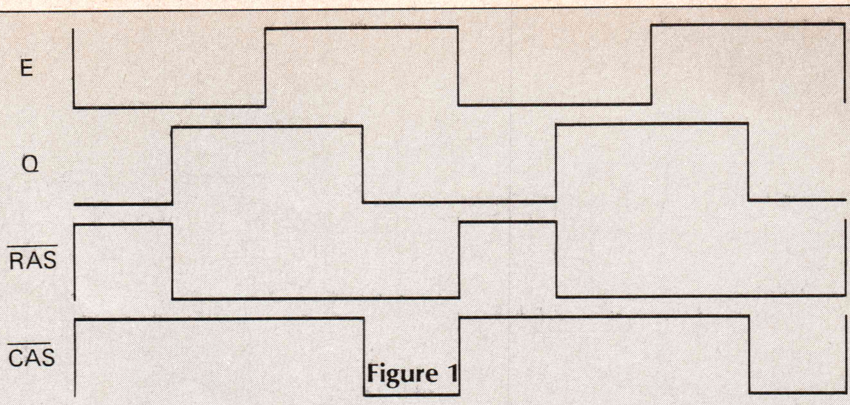


Figure 1

The timing diagram shows how the signals E and Q from the 6809 gate the column (CAS) and row (RAS) logic in the 6665 dynamic memory chips. This eliminates the need for delay circuits.

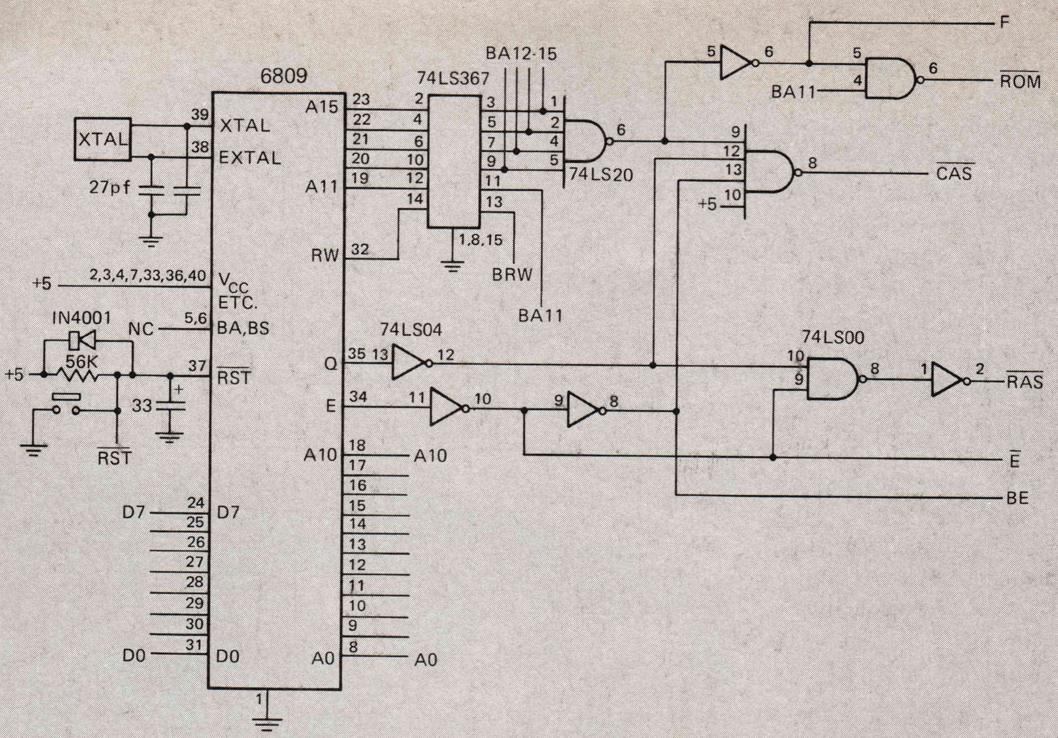


Figure 2.
The wiring diagram shows the 6809 and the address decoding. A 3–4 MHz crystal is required (frequency is not critical).

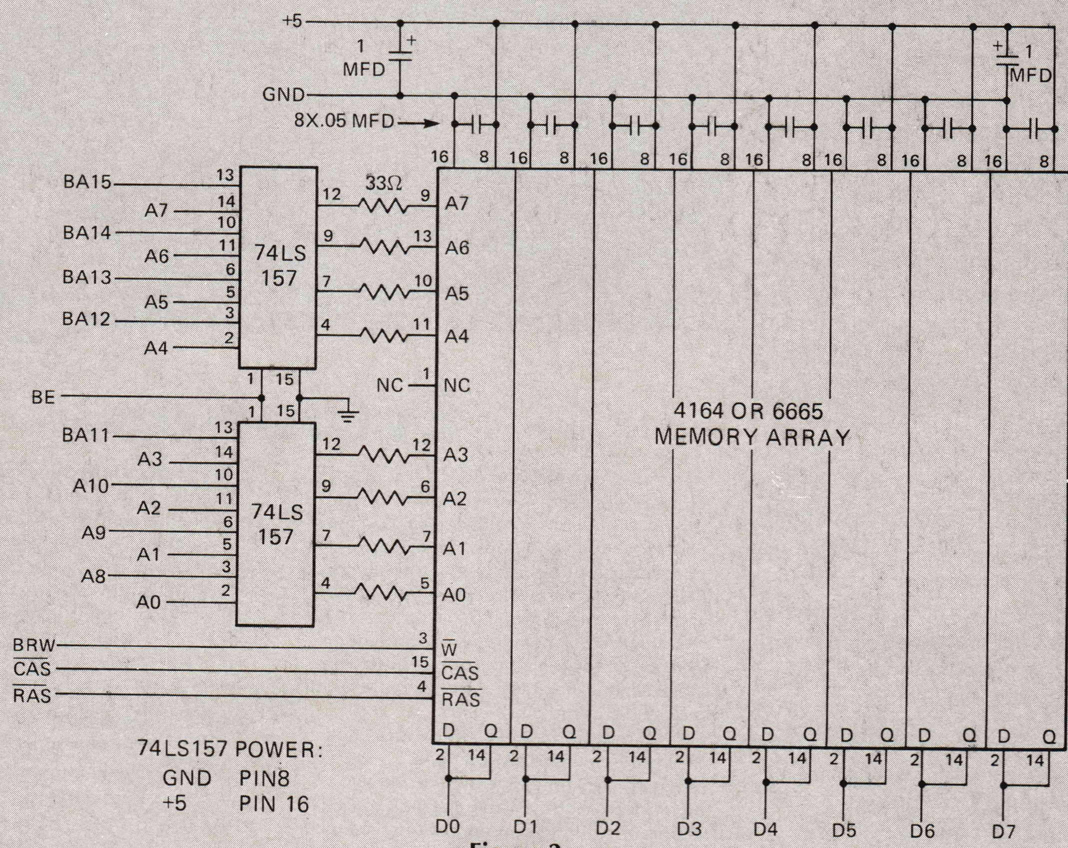


Figure 3.
The eight dynamic RAM chips are wired in parallel. Make identical connections to each chip with the exception of the data lines; a different data line connects to each chip as shown. Be sure to install a decoupling capacitor near each 6665.

PROGRAMMER DEVELOPMENT TOOLS

NEW LISTINGS:

	List	Ours
Brief By Solution Systems	195	**Call**
Megamax C compiler for MacIntosh	295	**239**
PC Lint by Gimpel Software	100	**89**
Prolog-86 by Solution Systems	125	**Call**
Scientific Subroutine Lib *for C* by Peerless	175	**159**

C-terp Complete C Interpreter
Full K&R C interpreter/semi-compiler which can access functions and externals compiled on various C compilers. Comes with a powerful, integrated screen editor providing a complete professional C programming environment.
List Price $300 Our Price **$269**

C LANGUAGE:

Computer Innovations C-86 Compiler	395	**299**
DeSmet C Compiler with Debugger	159	**145**
Lattice C Compiler from Lattice	500	**339**
Lattice C from Lifeboat *Ltd Qty Special*	500	**275**
Mark Williams C Compiler *w/Source Debugger*	495	**429**
Run/C Interpreter by Age of Reason *Sale*	150	**99**
Safe C Standalone Interpreter by Catalytix	Call	**Call**
Wizard C Compiler by Wizard Systems	450	**399**
Xenix Development System by SCO	1350	**1099**

Microsoft C Compiler version 3.0
This entirely new version of Microsoft's C compiler features fast execution and compact code generation, small, medium and large memory models, Xenix compatibility, a linker and a librarian.
List Price $395 Our Price **$339**

OTHER LANGUAGES:

8088 Assembler w/Z-80 Translator by 2500 AD	100	**89**
APL*PLUS/PC by STSC	595	**469**
BetterBASIC by Summit Software	200	**169**
Golden Common LISP by Gold Hill	495	**Call**
Janus/ADA by R&R Software	900	**699**
MASM-86 ver 3.0 *w/utilities* by Microsoft	150	**119**
Modula-2/86 by Logitech	495	**439**
Professional BASIC by Morgan Computing	99	**89**
RM/Fortran by Ryan-McFarland	595	**439**

C UTILITIES:

C Power Paks From Software Horizons	Call	**Call**
C-Sprite *Symbolic Debugger for Lattice C*	175	**159**
c-tree by FairCom	395	**359**
C Utility Library by Essential Software	Call	**Call**

C UTILITIES:

dBC *dBase/C Interface* by Lattice	250	**219**
dBC *with source code*	500	**459**
ESP for C by Bellesoft	Call	**Call**
GraphiC by Scientific Endeavors	250	**209**
Greenleaf C Functions Library *ver 3.0*	185	**139**
Greenleaf Comm Library	185	**139**
Multi-Halo Graphics by Media Cybernetics	250	**199**
PANEL *Screen Designer ver. 6.0* by Roundhill	295	**234**
Pasm86 *Macro Assembler* by Phoenix	295	**259**
Pfinish *Performance Analyzer* by Phoenix	395	**339**
Pmaker *Program Development Manager* by Phoenix	195	**179**
Pre-C *Lint Utility* by Phoenix	395	**339**
Safe C Dynamic Profiler by Catalytix	150	**Call**
Safe C Runtime Analyzer by Catalytix	400	**Call**
Windows For C by Creative Solutions	195	**139**

TURBO PASCAL:

Screen Sculptor by Software Bottling	125	**109**
Turbo ASYNCH by Blaise Computing	100	**89**
Turbo GRAPHICS TOOLBOX by Borland Int'l	55	**49**
Turbo PASCAL *ver 3.0* by Borland Int'l *Sale*	70	**49**
Turbo PASCAL *w/8087 or BCD* *Sale*	110	**89**
Turbo PASCAL *w/8087 & BCD* *Sale*	125	**99**
Turbo POWER TOOLS by Blaise Computing *New*	100	**89**
Turbo TOOLBOX by Borland Int'l	55	**49**
Turbo TUTOR by Borland Int'l	35	**29**
XTC Text Editor by Wendin	99	**89**

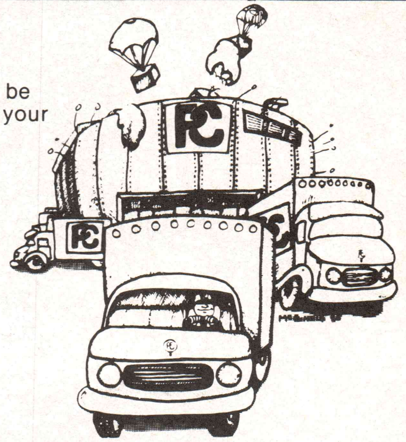
OTHER PRODUCTS:

Advanced Trace-86 by Morgan Computing	175	**159**
APL2C by Lauer Software *Interfaces APL to C*	150	**139**
Blaise Tools *for C & Pascal*	Call	**Call**
Btrieve by SoftCraft	250	**199**
Codesmith-86 *Debugger* by Visual Age	145	**129**
Epsilon *Emacs-like Text Editor* by Lugaru	195	**179**
FORTRAN Libraries by Alpha Computer Service	Call	**Call**
Pfix-86 Plus by Phoenix	395	**299**
Plink-86 *Overlay Linker* by Phoenix	395	**299**
Pmate *Macro Text Editor* by Phoenix	225	**159**
Polytron Products *We Carry a Full Line*	Call	**Call**
Profiler by DWB Associates	125	**89**
Xtrieve by SoftCraft	195	**169**

Periscope Symbolic Debugger
by Data Base Decisions
Write-protect memory board and breakout switch allows instant recovery from runaway code. Provides on-line help, windowing, extensive breakpoints, dual monitor support and more.
List Price $295 Our Price **$269**

ADVANTAGE #2

At Programmer's Connection most popular products are always in stock ready to be shipped. Most orders are on their way to you the same day they are placed. We know your time is important, that's why we tell you exactly when you can expect to receive your package. Maintaining an adequate inventory is part of our philosophy of fast, efficient service. Call us — you'll discover that Programmer's Connection delivers products and service without delay.



Discover the advantages of buying from Programmer's Connection:

1. We offer the latest version of a product.
2. Most popular products are in stock ready to be shipped.
3. Receive same manufacturer's support as if buying direct.
4. Experienced professional programmers are on staff.
5. Choose from a large selection of the best software products available.
6. Knowledgeable and courteous sales staff.
7. Significant discounts off of retail prices.
8. No extra charge on prepaid orders, including major credit cards.
9. Reasonable charges for shipping and handling.
10. Toll free services from Canada and the U.S.

"Programmers Serving Programmers"

Programmer's Connection
136 Sunnyside Street Hartville, Ohio 44632 (216) 877-3781 (In Ohio)
U.S.; **1-800-336-1166** Canada; **1-800-225-1166**
Call For Our Catalog

Account is charged when order is shipped
Prices are subject to change without notice

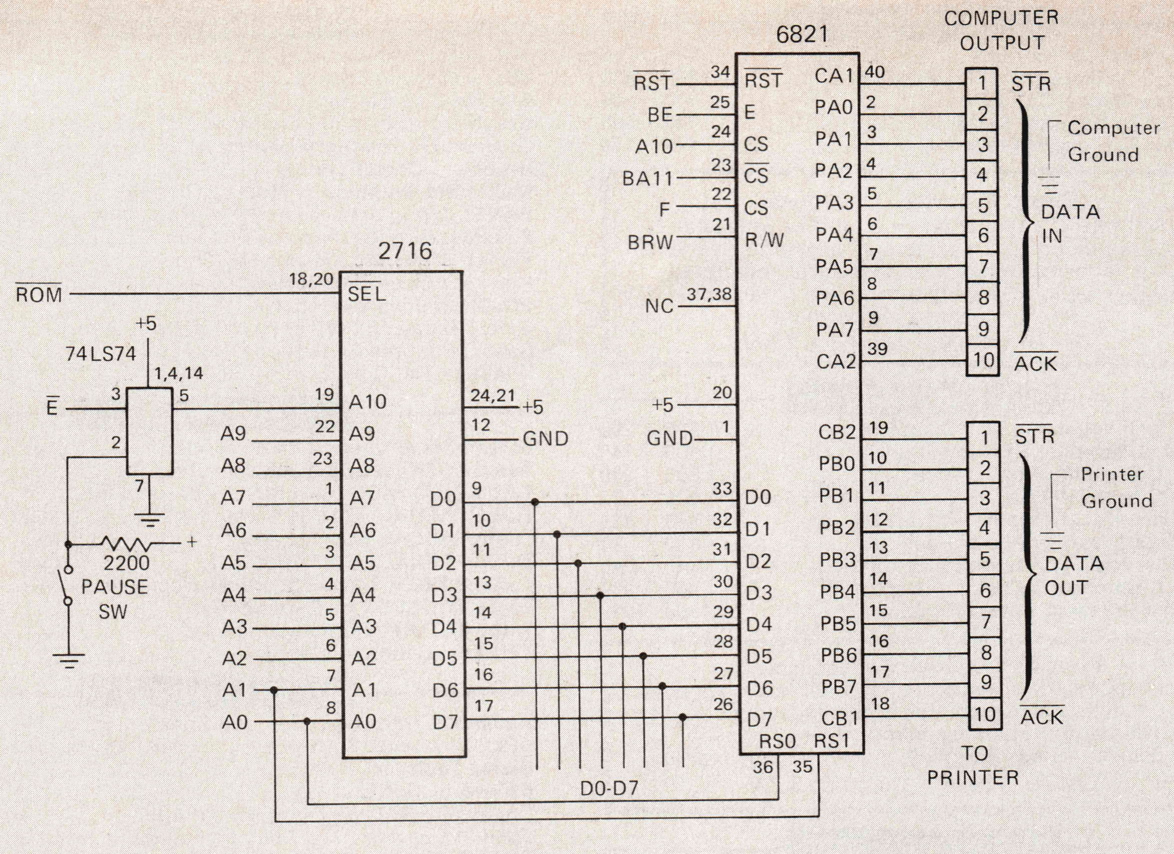


Figure 4.
Wiring of the 2716 EPROM and the 6821 parallel interface adapter.

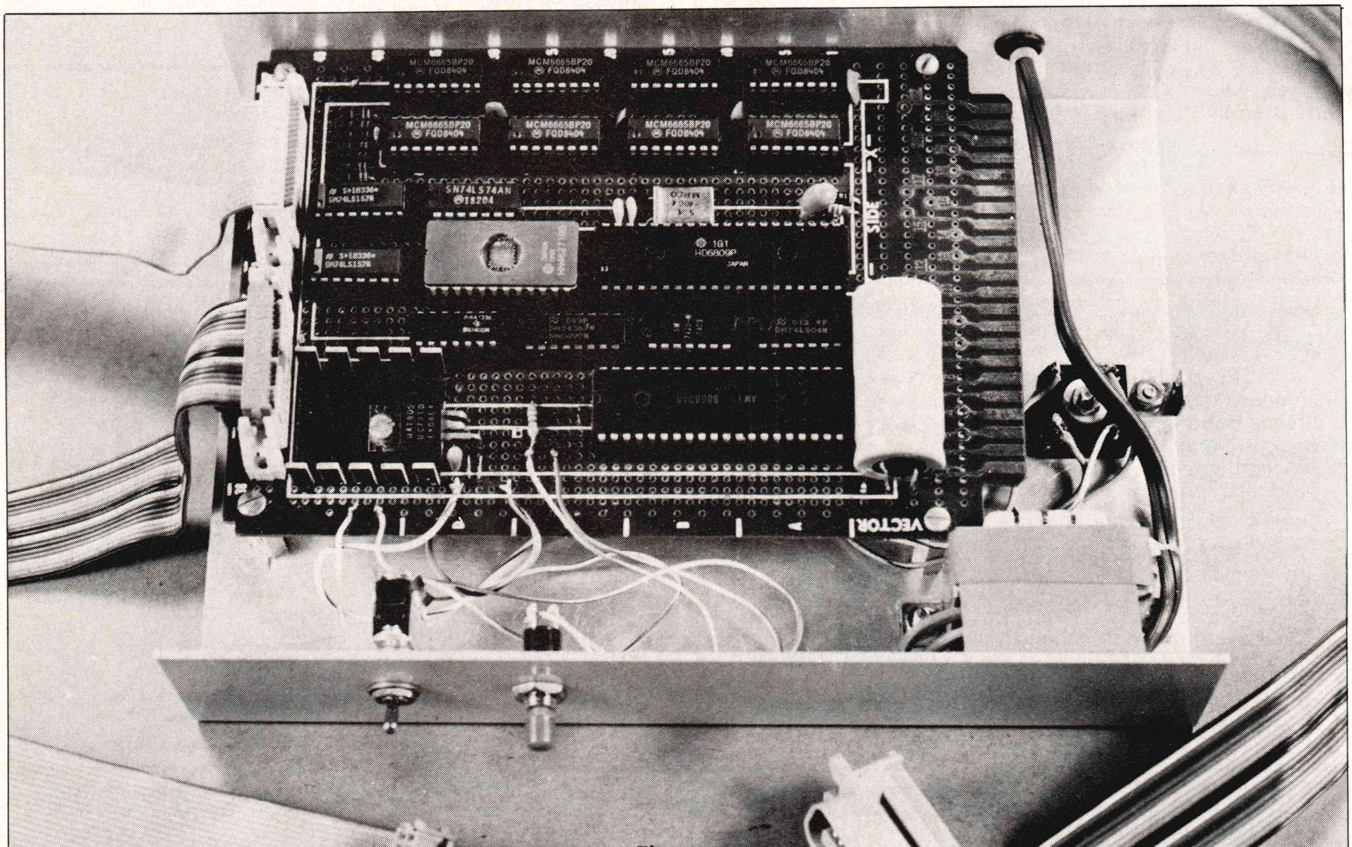


Figure 5.
Although the layout of the author's finished spooler is not critical, this photo can serve as a guide.

(1) The software must read every possible combination of the seven low-order addresses (A0–A6), at least one every 2 ms, to refresh the dynamic RAM.
(2) The software must bank-switch between the upper and lower halves of the ROM to provide the pause function.
Aside from this, the software is straightforward. The 6809 supports relocatable programs with its "long branch" instructions. Listing One (page 53) shows how these functions are implemented.

After a short routine to initialize the PIA, the program wakes up the printer with a null and sets up the X, Y, and U registers as pointers and counters. It next enters a loop that starts with a RAM refresh, which consists of stepping through 128 consecutive ADDA #0 instructions; this steps through our 128 consecutive addresses. The Motorola literature recommends 128 NOPs, which execute in 256μs, but I have elected to use 64 consecutive addresses in 128μs. During this sequence, once each byte, RAS on the dynamic RAMs is activated. Notice that the eight low-order addresses correspond to the 128 rows in the dynamic RAMs even though we are executing code reading the ROM. CAS remains high, and because the RAMs require both RAS and CAS for a read or write, the RAMs do not assert any data on the data bus. Refresh of the RAMs, however, does take place because the RAS signal alone sets off the refresh internally.

After the refresh sequence, the actual input, storage, and output of characters take place. It is critical that this routine not use any secondary loops that might prevent refresh. For example, if the printer is not ready, the routine continues on rather than waiting in a tight loop for it to get ready.

To implement a pause function with minimum hardware, we have put the same code in both halves of the ROM except for one instruction (see Listing Two, page 55): a BRA (branch always) operation in the active half of the ROM and a BRN (branch never) operation in the pause half. The change of a single bit ($20

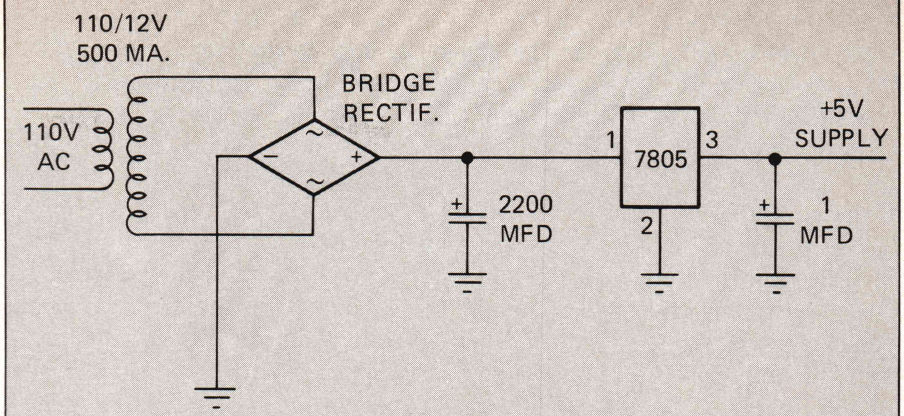


Figure 6.
The rectifiers, transformer, and switches are mounted in the enclosure, while the rest of the power-supply components fit on the circuit board (see Figure 5).

vs. $21) causes the output portion of the routine to be bypassed when you close the pause switch. See the parts list (page below) for a source for a preprogrammed ROM if you do not care to program your own 2716.

Construction

The only critical aspect of the spooler board's construction is to provide heavy power connections and good bypassing for the dynamic RAM chips. I used the Vector 3677-2 board, which has wide ground and 5 volt busses. This 4.5 × 6.5-inch board will accommodate the entire circuit, except for the transformer and rectifier, if you position the IC sockets carefully. If you are not sure, use the 4.5 × 9-inch board (Vector 3677).

The RAM bypass capacitors must be close to the power pins of the RAM chips. I used stripped wire-wrap throughout, rather than Just-wrap or equivalent, because my technique with the shortcut methods leaves an occasional bad connection, which is tedious to debug. Because we do not use the edge-connector, there is space to place the large power-supply capacitor on the board. The only off-board components are the power supply transformer, the rectifier, and the control switches.

Parts List

Integrated Circuits

1 MC6809 microprocessor (do not use MC6809E)
1 MC6821 peripheral interface adapter
8 MCM6665A or 4164 64K dynamic RAM 200 ns
2 74LS157 quad 2-line to 1-line data selector
1 74LS00 quad NAND gate
1 74LS04 hex inverter
1 74LS20 dual 4-input NAND gate
1 74LS367 hex bus driver
1 74LS74 dual D-type flip-flop
1 2716 EPROM, 5-volt type. Programmed version available at the Bit Stop, 5958 Shenandoah Rd., Mobile, AL 36608 ($30 ppd)
1 7805 5-volt regulator TO-220 case

Sockets and Plugs

2 40-pin wire-wrap sockets
1 24-pin wire-wrap sockets
11 16-pin wire-wrap sockets
4 14-pin wire-wrap sockets
2 20-pin wire-wrap header, JDR Microdevices IDH20W or equivalent
2 20-contact ribbon header sockets, JDR IDS20 or equivalent, plus 1–2 feet 20-conductor ribbon cable

Miscellaneous

1 Vector 3677-2 4.5 × 6.5-inch board or Vector 3677 4.5 × 9.0-inch board
1 4.0 MHz crystal or TV color-burst crystal
1 Heat sink for TO-220 regulator
1 1-amp 50 v. bridge rectifier
1 Power transformer, primary 110 v. secondary 12 v. at 500 ma
1 Reset pushbutton, normally open
1 Pause switch, SPST
1 2200 mfd 15 v. electrolytic cap.
1 33 mfd 15 v. trantalum cap.
2 27 pf. disc cap.
8 .05 mfd disc cap.
3 2 mfd 15 v. trantalum cap.
1 1N4001 diode
8 33 ohm ¼ w. resistor
1 56K ½ w. resistor
1 6 × 8 × 2-inch aluminum enclosure or larger (Radio Shack)

Most of the required parts are available from these alternate sources:
JDR Microdevices, 1224 S. Bascom Ave., San Jose, CA 95128 ordering line 1-800-538-5000; in CA 1-800-662-6279
Priority One Electronics, 9161 Deering Ave., Chatsworth, CA 91311 ordering line 1-800-423-5922
DoKay Computer Products, 2100 De La Cruz Blvd., Santa Clara, CA 95050 ordering line 1-800-538-8800; in CA 1-800-848-8008

Possible Design Modification

You can alter the design for use with a serial printer by replacing the PIA with a 6850 ACIA and adding a suitable baud-rate generator. The ACIA has some built-in handshaking capabilities that can control the interfacing of the host computer and printer.

As to memory refresh, you can divorce the refresh from the mainline routine by interrupting the microprocessor at 500 Hz or faster (2 ms). The interrupt routine would read your 128 consecutive addresses and then either return or do any other short function required before returning from interrupt.

By further decoding, you could increase the buffer size from 60K to about 63.5K, using only 0.5K for the ROM space and only a few bytes for the PIA. Note that we do not use the upper 4K of RAM in our design. Also note that all of the low RAM space is available for character buffering because the software uses internal microprocessor registers exclusively as pointers and counters.

Because the 60K spooler is a "smart" device with its own microprocessor, you could use routines to add a variety of control functions, such as automatic form feed. If your software gets complex, you may wish to implement a stack in RAM and call a subroutine for refresh, e.g.:

```
LOOP BSR REFRESH
     (test a flag)
BPL LOOP
     (continue)
```

Note that the loop contains a call to the refresh routine. Each loop where a significant delay is encountered should contain such a call.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 193.**

Print Spooler (Text begins on page 46)

Listing One

An assembled listing of the ROM contents. The 2K EPROM occupies locations $F8000 to $FFFF.

```
                     1000 * SPOOLER64 - DUAL
                     1010        .OR $F800
                     1020        .TA $0800
                     1030 *      .TF SPOOLER64.OBJ5
F3FF-                1040 STACK  .EQ $F3FF
F400-                1050 PORTA  .EQ $F400
F400-                1060 DDRA   .EQ $F400
F401-                1070 CRA    .EQ $F401
F402-                1080 PORTB  .EQ $F402
F402-                1090 DDRB   .EQ $F402
F403-                1100 CRB    .EQ $F403
0000-                1110 SOM    .EQ $0000
F000-                1120 EOM    .EQ $F000
F000-                1130 MAX    .EQ EOM-SOM
                     1140 *---------------------------------
F800-                1150        .BS $FB00-*  SKIP $0300 BYTES
                     1160 *---------------------------------
FB00- 7F F4 01       1170 START  CLR CRA       ACCESS DDR'S
FB03- 7F F4 03       1180        CLR CRB
FB06- 86 00          1190        LDA #$00      A - INPUT
FB08- B7 F4 00       1200        STA DDRA
FB0B- 86 FF          1210        LDA #$FF      B - OUTPUT
FB0D- B7 F4 02       1220        STA DDRB
FB10- 86 2F          1230        LDA #$2F      00101111
FB12- B7 F4 01       1240        STA CRA
FB15- 86 2F          1250        LDA #$2F      00101111
FB17- B7 F4 03       1260        STA CRB
FB1A- 86 00          1270        LDA #$00
FB1C- B7 F4 02       1280        STA PORTB     TOSS NULL
FB1F- B6 F4 00       1290        LDA PORTA     STROBE CA2
FB22- 8E 00 00       1300        LDX #SOM      X = INPTR
FB25- 10 8E 00
FB28- 00             1310        LDY #SOM      Y = OUTPTR
FB29- CE 00 00       1320        LDU #0        U = CHRCNT
FB2C- 10 CE F3
FB2F- FF             1330        LDS #STACK
                     1340 *---------------------------------
                     1350 * REFRSH IS HERE. CONSISTS OF
                     1360 * 64 ADDA #0 CODES FOR REFRESH.
                     1370 * INSTRUCTION IS 8B 00

                     1480 *---------------------------------
FBB0- 11 83 F0
FBB3- 00             1490 CHKIN  CMPU #MAX     CHECK COUNT
FBB4- 27 14          1500        BEQ PAUSE     SKIP IF MAX.
FBB6- F6 F4 01       1510        LDB CRA       GET INPUT FLAG
FBB9- 2A 0F          1520        BPL PAUSE     SKIP IF CLEAR
FBBB- B6 F4 00       1530        LDA PORTA     GET INPUT DATA
FBBE- A7 80          1540        STA ,X+       STORE & INCR X
FBC0- 33 41          1550        LEAU 1,U      INCR. COUNT
FBC2- 8C F0 00       1560        CMPX #EOM     END OF MEM?
FBC5- 26 03          1570        BNE PAUSE     BRANCH IF NOT
FBC7- 8E 00 00       1580        LDX #SOM      WRAPAROUND
FBCA- 21 03          1590 PAUSE  BRN CHKOUT    NEVER BRANCH
FBCC- 16 FF 61       1600        LBRA REFRSH
FBCF- 11 83 00
FBD2- 00             1610 CHKOUT CMPU #0       COUNT ZERO?
FBD3- 27 19          1620        BEQ GOREFR    SKIP IF 0
FBD5- F6 F4 03       1630        LDB CRB       GET PRTR READY
FBD8- 2A 14          1640        BPL GOREFR    BRANCH IF BUSY
FBDA- A6 A0          1650        LDA ,Y+       GET DATA. INCR Y
FBDC- B7 F4 02       1660        STA PORTB     PRINT IT
FBDF- B6 F4 02       1670        LDA PORTB     CLEAR IRQ FLAG
FBE2- 33 5F          1680        LEAU -1,U     DECR COUNT
FBE4- 10 8C F0
FBE7- 00             1690        CMPY #EOM     AT MEMORY END?
```

(Continued on next page)

Print Spooler
(Listing Continued, text begins on page 46)

Listing One

```
FBE8- 26 04     1700        BNE GOREFR    BRANCH IF NOT
FBEA- 10 8E 00
FBED- 00        1710        LDY #SOM      WRAPAROUND
FBEE- 16 FF 3F  1720 GOREFR LBRA REFRSH   GO TO REFRESH
                1730 *-----------------------------------
FBF1-           1740        .BS $FBFE-*   SPACE TO VECTORS
FBFE- FB 00     1750 RESET  .DA START     NEED ONLY RESET
                1760 *-----------------------------------
                1770 *-----------------------------------
FC00-           1780        .BS $FF00-*   SKIP $300 BYTES
                1790 *-----------------------------------
FF00- 7F F4 01  1800 STARTX CLR CRA       ACCESS DDR'S
FF03- 7F F4 03  1810        CLR CRB
FF06- 86 00     1820        LDA #$00      A - INPUT
FF08- B7 F4 00  1830        STA DDRA
FF0B- 86 FF     1840        LDA #$FF      B - OUTPUT
FF0D- B7 F4 02  1850        STA DDRB
FF10- 86 2F     1860        LDA #$2F      00101111
FF12- B7 F4 01  1870        STA CRA
FF15- 86 2F     1880        LDA #$2F      00101111
FF17- B7 F4 03  1890        STA CRB
FF1A- 86 00     1900        LDA #$00
FF1C- B7 F4 02  1910        STA PORTB     TOSS NULL
FF1F- B6 F4 00  1920        LDA PORTA     STROBE CA2
FF22- 8E 00 00  1930        LDX #SOM      X = INPTR
FF25- 10 8E 00
FF28- 00        1940        LDY #SOM      Y = OUTPTR
FF29- CE 00 00  1950        LDU #0        U = CHRCNT
FF2C- 10 CE F3
FF2F- FF        1960        LDS #STACK
                1970 *-----------------------------------
                1980 * REFRSH IS HERE. CONSISTS OF
                1990 * 64 ADDA #0 CODES FOR REFRESH.
                2000 * INSTRUCTION IS 8B 00

                2110 *-----------------------------------
FFB0- 11 83 F0
FFB3- 00        2120 CHKINX CMPU #MAX     CHECK COUNT
FFB4- 27 14     2130        BEQ PAUSEX    SKIP IF MAX.
FFB6- F6 F4 01  2140        LDB CRA       GET INPUT FLAG
FFB9- 2A 0F     2150        BPL PAUSEX    SKIP IF CLEAR
FFBB- B6 F4 00  2160        LDA PORTA     GET INPUT DATA
FFBE- A7 80     2170        STA ,X+       STORE & INCR X
FFC0- 33 41     2180        LEAU 1,U      INCR. COUNT
FFC2- 8C F0 00  2190        CMPX #EOM     END OF MEM?
FFC5- 26 03     2200        BNE PAUSEX    BRANCH IF NOT
FFC7- 8E 00 00  2210        LDX #SOM      WRAPAROUND
FFCA- 20 03     2220 PAUSEX BRA CHKOUX
FFCC- 16 FF 61  2230        LBRA REFRSX
FFCF- 11 83 00
FFD2- 00        2240 CHKOUX CMPU #0       COUNT ZERO?
FFD3- 27 19     2250        BEQ GOREFX    SKIP IF 0
FFD5- F6 F4 03  2260        LDB CRB       GET PRTR READY
FFD8- 2A 14     2270        BPL GOREFX    BRANCH IF BUSY
FFDA- A6 A0     2280        LDA ,Y+       GET DATA. INCR Y
FFDC- B7 F4 02  2290        STA PORTB     PRINT IT
FFDF- B6 F4 02  2300        LDA PORTB     CLEAR IRQ FLAG
FFE2- 33 5F     2310        LEAU -1,U     DECR COUNT
FFE4- 10 8C F0
FFE7- 00        2320        CMPY #EOM     AT MEMORY END?
FFE8- 26 04     2330        BNE GOREFX    BRANCH IF NOT
FFEA- 10 8E 00
FFED- 00        2340        LDY #SOM      WRAPAROUND
FFEE- 16 FF 3F  2350 GOREFX LBRA REFRSX   GO TO REFRESH
                2360 *-----------------------------------
FFF1-           2370        .BS $FFFE-*   SPACE TO VECTORS
FFFE- FF 00     2380 RESETX .DA STARTX    NEED ONLY RESET
```

End Listing One

Listing Two

A hexadecimal dump of the EPROM. Base address for this dump is $0800 (corresponds to the first byte of the EPROM).

```
>$0B00.0BFF

0B00- 7F F4 01 7F F4 03 86 00
0B08- B7 F4 00 86 FF B7 F4 02
0B10- 86 2F B7 F4 01 86 2F B7
0B18- F4 03 86 00 B7 F4 02 B6
0B20- F4 00 8E 00 00 10 8E 00
0B28- 00 CE 00 00 10 CE F3 FF
0B30- 8B 00 8B 00 8B 00 8B 00
0B38- 8B 00 8B 00 8B 00 8B 00
0B40- 8B 00 8B 00 8B 00 8B 00
0B48- 8B 00 8B 00 8B 00 8B 00
0B50- 8B 00 8B 00 8B 00 8B 00
0B58- 8B 00 8B 00 8B 00 8B 00
0B60- 8B 00 8B 00 8B 00 8B 00
0B68- 8B 00 8B 00 8B 00 8B 00
0B70- 8B 00 8B 00 8B 00 8B 00
0B78- 8B 00 8B 00 8B 00 8B 00
0B80- 8B 00 8B 00 8B 00 8B 00
0B88- 8B 00 8B 00 8B 00 8B 00
0B90- 8B 00 8B 00 8B 00 8B 00
0B98- 8B 00 8B 00 8B 00 8B 00
0BA0- 8B 00 8B 00 8B 00 8B 00
0BA8- 8B 00 8B 00 8B 00 8B 00
0BB0- 11 83 F0 00 27 14 F6 F4
0BB8- 01 2A 0F B6 F4 00 A7 80
0BC0- 33 41 8C F0 00 26 03 8E
0BC8- 00 00 21 03 16 FF 61 11
0BD0- 83 00 00 27 19 F6 F4 03
0BD8- 2A 14 A6 A0 B7 F4 02 B6
0BE0- F4 02 33 5F 10 8C F0 00
0BE8- 26 04 10 8E 00 00 16 FF
0BF0- 3F 00 00 00 00 00 00 00
0BF8- 00 00 00 00 00 00 FB 00
```

```
>$0F00.0FFF

0F00- 7F F4 01 7F F4 03 86 00
0F08- B7 F4 00 86 FF B7 F4 02
0F10- 86 2F B7 F4 01 86 2F B7
0F18- F4 03 86 00 B7 F4 02 B6
0F20- F4 00 8E 00 00 10 8E 00
0F28- 00 CE 00 00 10 CE F3 FF
0F30- 8B 00 8B 00 8B 00 8B 00
0F38- 8B 00 8B 00 8B 00 8B 00
0F40- 8B 00 8B 00 8B 00 8B 00
0F48- 8B 00 8B 00 8B 00 8B 00
0F50- 8B 00 8B 00 8B 00 8B 00
0F58- 8B 00 8B 00 8B 00 8B 00
0F60- 8B 00 8B 00 8B 00 8B 00
0F68- 8B 00 8B 00 8B 00 8B 00
0F70- 8B 00 8B 00 8B 00 8B 00
0F78- 8B 00 8B 00 8B 00 8B 00
0F80- 8B 00 8B 00 8B 00 8B 00
0F88- 8B 00 8B 00 8B 00 8B 00
0F90- 8B 00 8B 00 8B 00 8B 00
0F98- 8B 00 8B 00 8B 00 8B 00
0FA0- 8B 00 8B 00 8B 00 8B 00
0FA8- 8B 00 8B 00 8B 00 8B 00
0FB0- 11 83 F0 00 27 14 F6 F4
0FB8- 01 2A 0F B6 F4 00 A7 80
0FC0- 33 41 8C F0 00 26 03 8E
0FC8- 00 00 20 03 16 FF 61 11
0FD0- 83 00 00 27 19 F6 F4 03
0FD8- 2A 14 A6 A0 B7 F4 02 B6
0FE0- F4 02 33 5F 10 8C F0 00
0FE8- 26 04 10 8E 00 00 16 FF
0FF0- 3F 00 00 00 00 00 00 00
0FF8- 00 00 00 00 00 00 FF 00
```

End Listing Two

Designing a Real-Time Clock for the S-100 Bus

by Alan D. Wilcox

The author, with time on his hands and mind, builds a hardware clock for his CompuPro 816/Z.

After revising my latest program and printing it out, I set it aside to look over later. Naturally, it was in a stack of earlier versions of the program that didn't quite work right and looked almost the same except for a few hard-to-find bugs. You know the rest of the story: when I came back in a few weeks, I couldn't tell which was the latest and therefore correct copy of my program. If I had jotted down the time or at least the date on my latest copy, I could have identified it then. So I decided it was time to design a clock circuit to do the writing for me automatically.

The result is a clock circuit using the popular National MM58167A real-time clock IC. The circuit requires no changes in the operating system and has battery backup so you need not reset it after turning off the computer. Designed to meet S-100/IEEE-696, it performs easily with virtually any S-100 microprocessor system. Two assembly language programs set and print the time and date; a third program prints the time, date, and filename as a title header then prints out the file in 60-line pages.

Background

There are two basic ways to put a clock in a computer: either modify the system software to keep track of the time or provide a piece of hardware that will keep time independently of the computer. The software approach is easier in that you need not tamper with the computer hardware, but it has several drawbacks. First, when power is off, the clock is off; this requires resetting the clock every time you turn the computer on. Second, you must modify the *system* software to support a time-keeping program; procedures for this modification can be almost incomprehensible to the novice. Finally, the subsequent time-keeping isn't accurate anyway; for example, the clock must be off during disk read or write operations. The hardware approach avoids these problems.

The concept of a hardware clock is simply to have a clock available for the computer to read when it wants the time and date. The clock has its own battery power supply to keep it running when the computer is turned off. The only programming required is for setting the clock and reading it; these straightforward programs do not require an intimate understanding of the system software. Finally, accuracy is crystal controlled and is unaffected by other computer operations.

A number of articles have discussed how to implement a clock in a personal computer. In one design, Calaway and Hill presented what I call a software approach to

Alan D. Wilcox, 60 South Eighth St., Lewisburg, PA 17837.

time-keeping.[1] Their basic idea is to provide a small amount of hardware to generate an interrupt each second; their system responds to this interrupt by updating the total number of pulses counted and thereby updating the clock time. The operating system needs modification, however, which can range from straightforward to profoundly complex.

Hassebrock described a simple approach to keeping the time and date using the Hayes Chronograph.[2] The Hayes unit is a piece of hardware connected to the computer through a serial data port. When you want the time or date, you run a program that tells the clock to send the information to the computer for printing or display. Because the Chronograph is connected to a serial port, you must configure the port to match the required data rate, and so on. Therefore, you need a modem-type program to access the clock; the program may be either part of the time-setting/reading program or part of the operating system. Hassebrock chose the latter and modified his system somewhat. Overall, the concept of an external serial-data clock is appealing except that a spare serial port may not be available.

A hardware approach described by Ciarcia seemed initially to offer some distinct advantages.[3] First, the design uses hardware to keep time: no difficulty keeping time when the power is off, no system software modifications, and no accuracy problems. Second, the clock does not use a serial port, making port initialization unnecessary. The intended application was for use with the Z8-BASIC microcomputer as an intelligent clock. Although the approach has merit, as it stands, it is not a generally useful design.

Design Overview

The design requirement was for a real-time clock to operate in my CompuPro System 816/Z, a Z80B S-100 machine running at 6 MHz. The clock should keep both the time and date, as well as provide a means of interrupting the system either on a regular basis or at a predetermined time. It should maintain an accuracy within 10 sec/month, with or without system power. Except for initially setting the time and date, the operator should not have to interact with the clock in any way.

The clock should meet S-100/IEEE-696 and operate as a slave without affecting normal system operation; likewise, it should require no changes in the system software. Data transfer between clock and system should use I/O-mapped ports. The handshaking should use the S-100 RDY line, and because the processor operates at 6 MHz, some wait states should be available. Clock interrupts to the system should be switchable to any of the S-100 vectored interrupt lines.

Although all initial programs should poll the clock for the time, the clock should be able to use any of the vectored interrupt lines to implement a future real-time multitasking executive; in the long term, this could involve multiprocessing with a new 10 MHz 68000 CPU board in the system. For the near term, however, the software should at least set and display the time, as well as print files with a time/date header.

Hardware Design

The National MM58167A real-time clock IC[4] provided a solution to the design requirements. As shown in the block diagram (Figure 1, page 58), you can partition hardware using this clock IC into several major modules that include the clock, address decoder, data-bus interface, interrupt switches, and power supply. Once you divide the hardware into modules, the design can proceed in much the same way as in software development: top down, bottom up, or most critical first.

The appropriate design method in this case is to consider the most critical section first. Everything depends on the clock IC, and its requirements come before all else. The block diagram of the MM58167A (Figure 2, page 59) shows that the IC uses five address lines (32 different addresses) and a chip select and has a single bidirectional data bus with separate read/write controls, two interrupt outputs, and a power-down control. Designing to accommodate each of these requirements individually results in the implementation shown in the circuit schematic (Figure 3, page 60).

I decided to use I/O mapping rather than memory mapping for the clock data transfer. With a maximum of 256 ports, only the lower eight bits of the address bus need decoding, and five of these are decoded internally by the clock IC. For maximum flexibility, I chose to use DIP switches to select the three most significant bits of the I/O address. The output of the decoder is used as a chip select for the clock and by the read/write qualifier circuit.

The S-100 data bus in and out lines are buffered with a pair of 74LS244s connected directly to the clock IC. The buffer for data-in (DI from the processor's viewpoint) is strobed using the I/O read qualifiers pDBIN and sINP and the chip select CS. The data-out buffer is strobed using pWR*, sOUT, and the chip select CS. During a typical read bus cycle, for example, the system places the proper address on the address bus, asserts sINP, then asserts pDBIN to strobe the DI buffer and the clock RD* input. In a typical write bus cycle, the system puts the address on the bus, asserts sOUT then asserts pWR* to strobe the DO buffer and the clock WR* input.

Timing

A basic read or write bus cycle has three bus states (BS1, BS2, and BS3), each of which takes 167 ns in a 6 MHz system; consequently, a bus read or write normally completes within 500 ns. However, for I/O devices that require substantial access times, you can extend the bus cycle by adding a number of wait states. To determine whether wait states are required during any read or write bus cycle, the bus master samples the RDY line at the rising edge of the system clock in BS2. If the RDY line is low, then a wait state (BSw) is inserted immediately after BS2; if the RDY line is still low one bus state later, yet another BSw is inserted. The addition of wait states continues until RDY finally goes high; at that point, the bus cycle concludes with BS3.

The MM58167A clock requires the addition of wait states because of its slow access time. For a clock read

operation, the maximum specified time from valid address until valid data is 1050 ns, far longer than a normal read bus cycle with no wait states. The read bus cycle on my 6 MHz CompuPro system is shown in Figure 4 (page 62) with approximate times to scale. Asserting pDBIN at the beginning of BS2 leaves no time for the clock to bring the RDY line low. However, all Z80 microprocessors automatically insert a single wait state for I/O instructions (by carrying out a microcoded instruction in the Z80), so the clock really has until the beginning of the first wait state BSw (auto) to get RDY low.

The timing question becomes: "Can clock RD* be pulled low and can the clock respond with RDY low—all within 167 ns?" The specs for the clock indicate that read-strobe to ready-strobe (clock pin 4) is 150 ns maximum, but the 7406 easily could add another 40 ns in propagation delay to the system RDY line. Add these up. In the worst case, the clock board cannot get the RDY line down in time even by taking advantage of the Z80 automatic wait state. As indicated in Figure 4, the clock actually takes from about 100 ns (middle of BS2) to about 190 ns (middle of BSw). Actual timing measurements and read operations indicate that the clock sometimes is fast enough but usually is not. For reliable operation of RDY, we must add an extra wait state.

In my system, I used the Z80 processor board's option switch to add a single wait state to I/O operations in addition to the Z80 automatic wait. This required no additional hardware, although you could enhance a more general clock design by including an on-board wait state generator circuit. Libes and Garetz present a number of suitable circuits in their book.[6]

When the clock data is finally valid, the system allows the RDY line to go high, and the processor reads the data in BS3. To finish BS3, pDBIN is negated, which raises the clock RD* input, and the address is deselected.

The write bus cycle timing requirements for the clock are similar to the read timing requirements. As in the read, you must include a single wait state to allow sufficient time for RDY to be pulled low during the write operation.

Power-Down Design

The power-down design is critical in the actual operation of the clock. If POWER DOWN* is not asserted to a low-

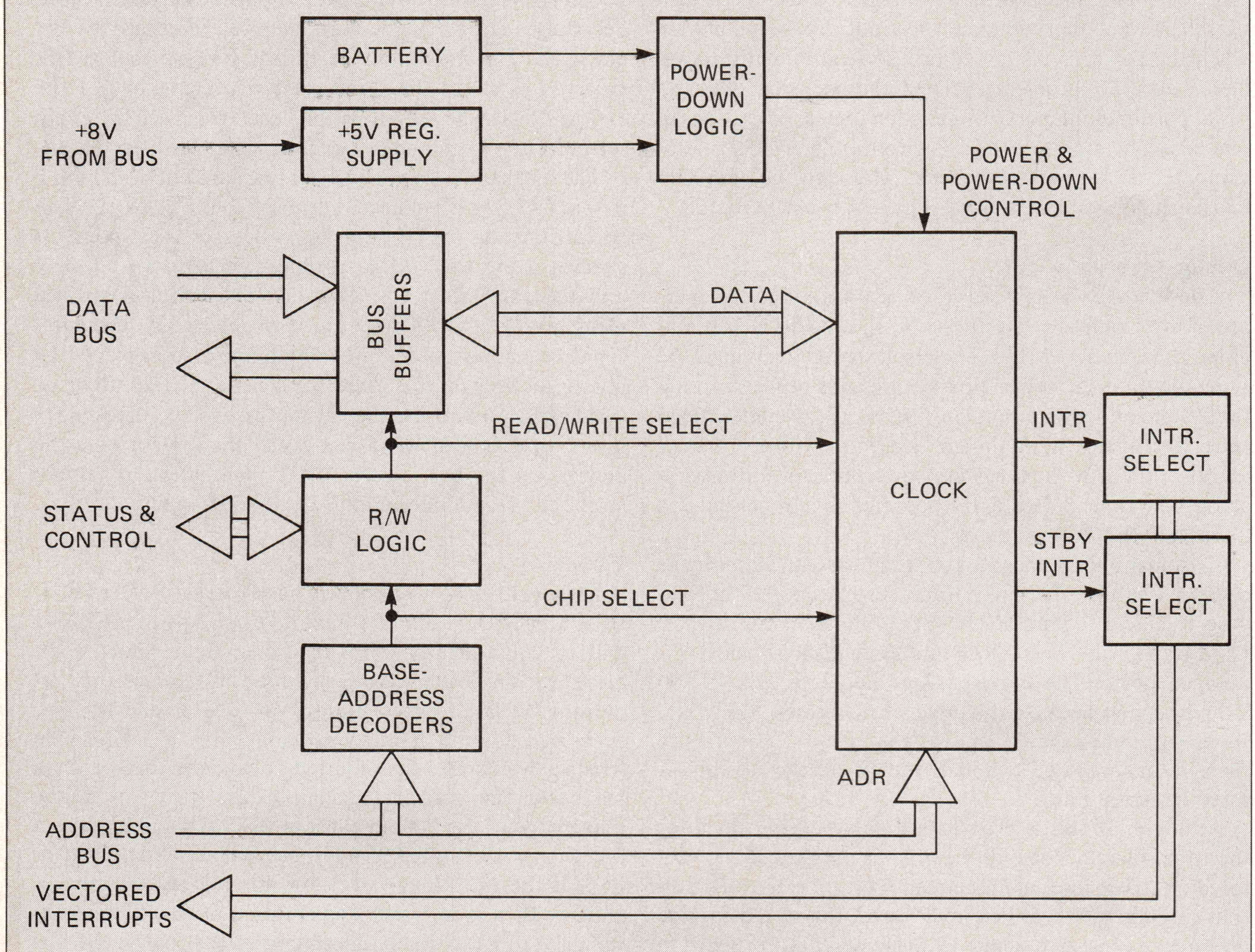


Figure 1.
Block Diagram of Real-Time Clock Board

voltage level at least one microsecond before system power removal, then the contents of the clock memory might be anything (or nothing) when you restore normal system power. Furthermore, when the power does come back on, POWER DOWN* must be held low until all bus signals are valid.

A common way of accomplishing this is to use a zener diode that sets a reference threshold voltage to control a transistor switch. Consider the circuit (Figure 3) when the system power is off: Q1 and Q2 are both off. When the system power comes on, nothing happens until the system bus voltage reaches about 6.6 V. Then the zener conducts, turning on Q2, which then turns on Q1. When Q1 goes on, voltage is applied to POWER DOWN*, which activates the clock chip for normal operation. Because a system bus voltage greater than 6.6 V is enough to allow near-normal output from all 5 V regulators, all bus signals should be valid by this time. When the system power goes off, the system voltage drops gradually enough so that, when it passes below 6.6 V, the zener and Q1/Q2 can drop the POWER DOWN* to a low voltage before the bus signals are no longer valid.

A small, 3.75 V, 20 mAh, NiCad rechargeable battery maintains clock operation when system power is off. Specified current consumption of the clock is on the order of 10 to 20 uA during power-down; my clock measured 14 uA while using the battery with a series-blocking diode. The 470 ohm resistor provides a trickle charge from ½ mA to about 2 mA, depending on the state of the battery and component tolerances. For an average use of several hours a day, this resistor keeps the battery charged between 3.8 and 4.15 V with no difficulties.

Logic Circuits

IEEE Std-696 (sec. 3.7) states that a card may not source more than 0.5 mA at 0.5 V nor sink more than 80 uA at 2.4 V on most system signal lines.[5] The effect of this is that you can connect only one LS TTL gate per card to each line of the address bus. My decoder uses the 74LS136 exclusive-or, which sources a worst-case maximum of 0.8 mA at 0.4 V. I expect this to cause no operational problems in any practical sense. I opted to use the 74LS136, rather than buffer the address bus with extra gates, to be strictly within the specification. I prefer to be conscious of trade-offs than to not be aware of them and have a problem occur later.

The pull-up resistors for the open-collector 7406 are selected to maintain proper logic levels and currents. The design trade-off is to let the various resistors be high for lower current consumption or to let them be low for more speed. I designed for as much speed as possible. Typical calculations are discussed in the standard *TTL Data Book* (page 6-6).[7]

Near the end of the project, I decided to implement the clock standby interrupt, which required an extra non-inverting gate. I used the spare 74LS10 and the last spare

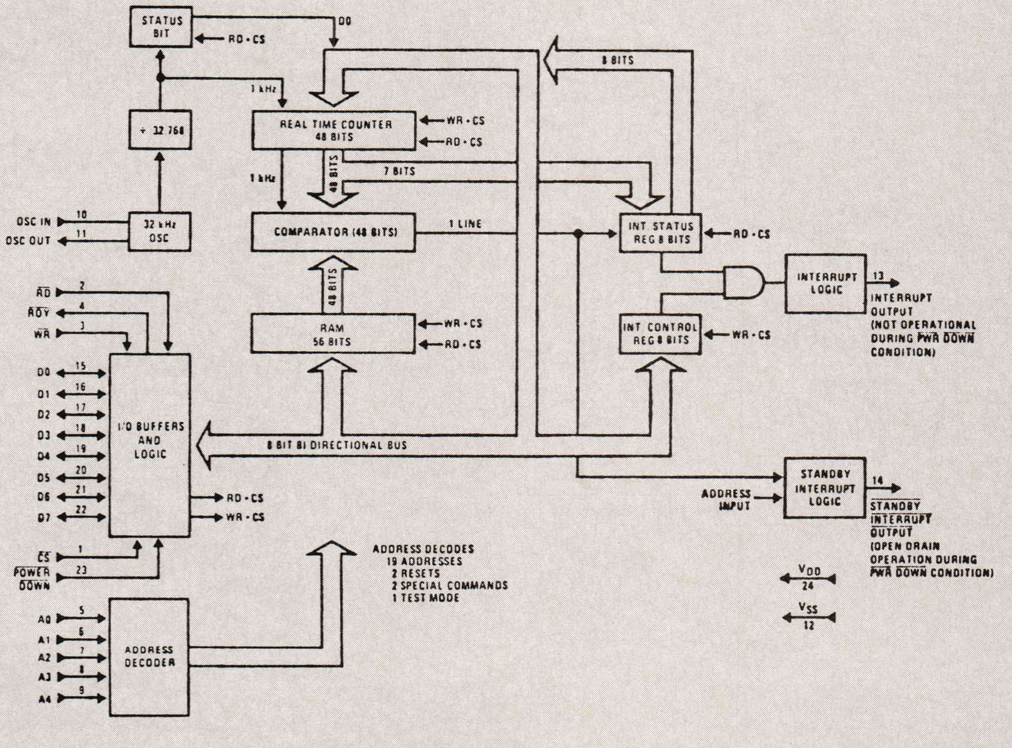
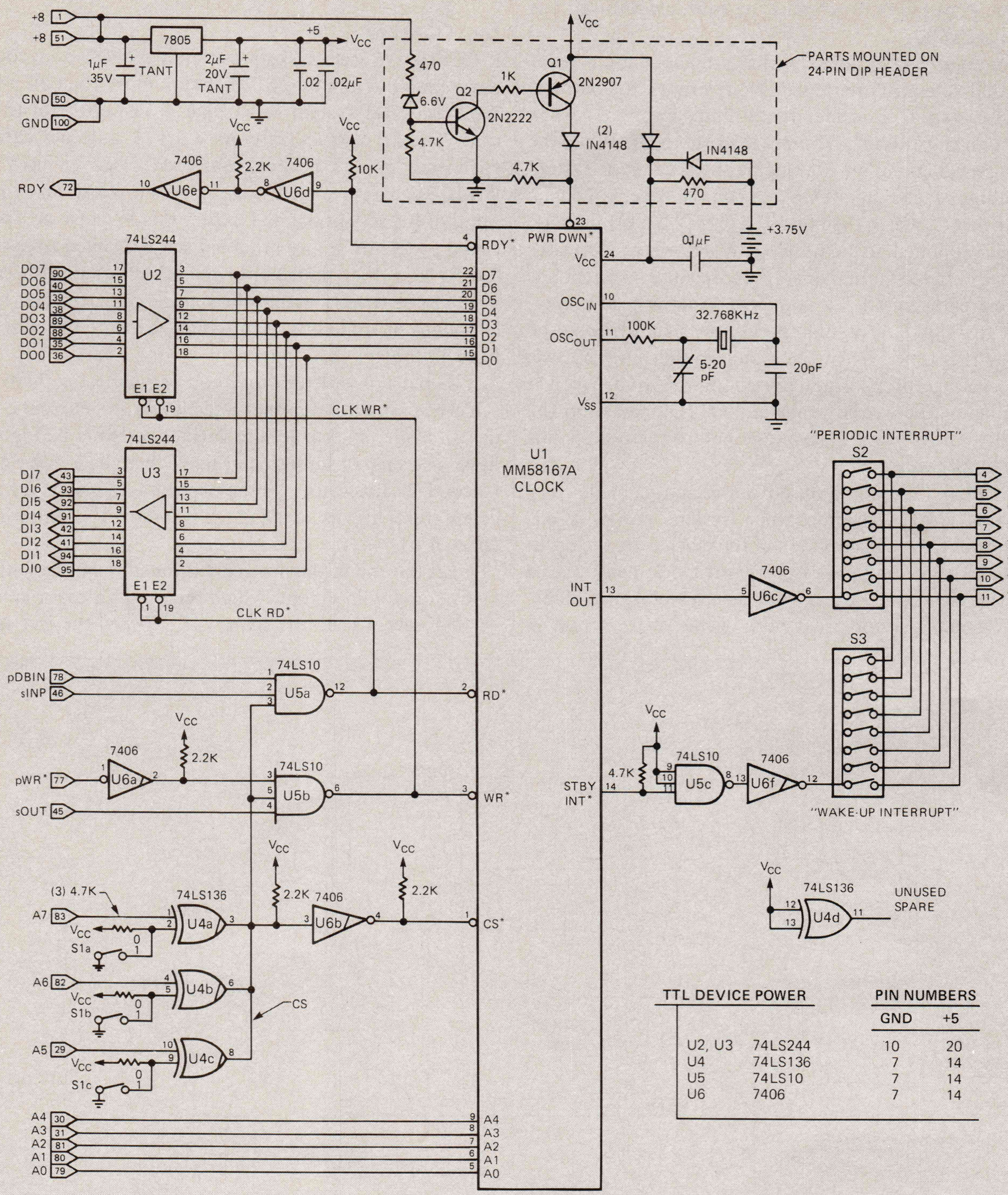


Figure 2.
Block Diagram of the MM58167A (material courtesy of National Semiconductor Corporation)



TTL DEVICE POWER		PIN NUMBERS GND	+5
U2, U3	74LS244	10	20
U4	74LS136	7	14
U5	74LS10	7	14
U6	7406	7	14

*NOTE: Leave interrupt switches S2 and S3 all open unless interrupts are to be implemented. If interrupts are used, close only 1 switch in S2 and 1 switch in S3 at most.

Figure 3.
Schematic Wiring Diagram of S-100 Real-Time Clock

DE SMET C

8086/8088 Development Package

$109

FULL DEVELOPMENT PACKAGE

- Full K&R C Compiler
- Assembler, Linker & Librarian
- Full Screen Editor
- Execution Profiler
- Complete STDIO Library (>120 Func)

Automatic DOS 1.X/2.X SUPPORT

BOTH 8087 & S/W FLOATING POINT

OVERLAYS

OUTSTANDING PERFORMANCE

- First and Second in AUG '83 BYTE benchmarks

SYMBOLIC DEBUGGER $50

- Examine & change variables by name using C expressions
- Flip between debug and display screen
- Display C source during execution
- Set multiple breakpoints by function or line number

DOS LINK SUPPORT $35

- Converts DeSmet.O to DOS.OBJ Format
- LINKs with DOS ASM
- Uses Lattice® naming conventions

C WARE CORPORATION

P.O. Box C, Sunnyvale, CA 94087

(408) 720-9696

Street Address: 505 W. Olive, #767 (94086) Call for hrs.

All orders shipped UPS surface on IBM format disks. Shipping included in price. California residents add sales tax. Canada shipping add $5, elsewhere add $15. Checks must be on U.S. Bank and in U.S. Dollars. Call 9am–1pm to CHARGE by VISA/MC/AMEX.

Foreign Distributors: **AFRICA,** HI-TECH SVCS, Gaborone 4540 or Telex 2205BD LANGER • **ENGLAND:** MLH Tech, 0606-891146 • **JAPAN:** JSE 03-486-7151 • **SWEDEN:** ESCORT DATA 08-87 41 48 **or** THESEUS KONSULT 08-23 61 60

Circle **no. 18** on reader service card.

ConIX™

NOW ONLY $79.95!

If you think you're missing out on innovative software developments because nobody is writing for CP/M™-80, take a look at us. We've adapted UNIX™ features to CP/M like never before, and with the kind of professional, quality-controlled product that you deserve. That product is none other than the critically acclaimed ConIX Operating System.

ConIX can provide any 48K+ CP/M-80 or compatible system with I/O Redirection and Pipes (uses memory or disk), perfected User Areas, Command and Overlay Path Searching, Auto Screen Paging, 8Mb Print Buffering, 22 new SysCalls, Function Keys, "Virtual" disk system, Archiver (saves over 50% disk), extensive command language, 300+ variables, 100+ commands, pull-down menu, and much more! Uses as little as 1/2K RAM! Runs with CP/M for true data and software compatibility. Installs easily without any system mods!

The ConIX package lists at $165 and has been advertised and sold internationally to many enthusiastic customers since October 1983. As a special limited offer, we've lowered the price of the complete ConIX system by 50% to only $79.95! Don't miss this opportunity to bring your 8-bit micro back into the software revolution. Order your copy of ConIX today!

Price includes manual, 8" disk, and user support. 5¼" conversions available. Contact your local dealer, or buy direct and add shipping: $4.50 UPS, $10 Canada, $25 overseas. NY residents add sales tax.

Computer Helper Industries Inc.
P.O. Box 680 Parkchester Station, NY 10462
Tel. (212) 652-1786 (for information/orders)

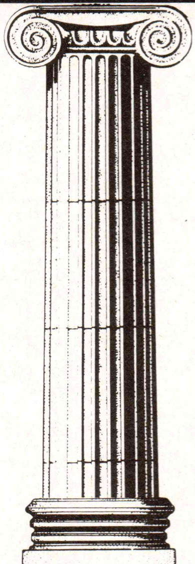
"We're helping your computer work better for you!"

UNIX: AT&T Bell Labs, CP/M: Digital Research, ConIX: Computer Helper Ind.

Circle **no. 22** on reader service card.

THE SYMBOLIC *SINE QUA NON*
L I S P

THE fifth generation language
P R O L O G



Cybermetrics UNXLISP-86 (tm) Features:

- 1 Megabyte address space.
- LAMBDA, NLAMBDA, MACRO with displacement.
- CATCH, THROW, ERRSET, & ERR.
- SAVE/RESTORE virtual images to/from disk.
- &OPTIONAL, &REST, and &AUX parameters.
- Tree-structured object list, functional directories.
- Lisp utilities with source, including pretty-printer, structure editor, debugging functions.
- Works with any text editor through exec function.
- Extensive manual – written in ENGLISH
- Requires 320K memory and runs under MSDOS and PCDOS versions 2.0 and later.

TECHNICAL
(408) 725-1344

$49.00

ADD $3.00 FOR
SHIPPING & HANDLING

Implementing the full Edinburgh Syntax as described by Clocksin and Mellish.

Recognized by Japan as providing unparalleled opportunity for artificial intelligence.

Applications:

- The highest level of a hierarchial robotic control system.
- Machine recognition of natural language.
- Expert systems and knowledge engineering.

Optional:

- Virtual memory
- Special libraries
- Language extensions
- Large model

Requires 192K memory and runs under MSDOS and PCDOS versions 2.0 and later.

TECHNICAL
(215) 646-4894

educational package
$29.95
other versions $50–$500

ADA automata design assoc.

1570 ARRAN WAY
DRESHER, PA 19025
TO ORDER CALL **(215) 355-5400** USE IT FOR 30 DAYS

VISA
MASTERCARD

Circle **no. 130** on reader service card.

7406 gate to get this output from the clock board. This particular output does not have the flexibility of the programmable output, but you can use it to indicate a match between real time and a preset time for a wake-up alarm.

Construction and Installation

I prototyped the entire real-time clock on a Vector 8800V wire-wrap board, as shown in the photo (page 64). I added the interrupt switches (S3) for the wake-up interrupt after taking the photo; they are located beside the main interrupt switches (S2). All the circuits took less than a third of a standard S-100 board. I wired the transistor power-down circuit on the 24-pin header at the lower-left edge of the board by the battery. I used plastic "wrap ID" panels on the back of the board for each IC socket to make component and pin identification easier.

My system does not have any I/O port assignments in the range A0h to BFh, so I selected A0 as the base address for the clock. For this base address, S1 sections a and c are closed for a "1" and section b is left open for a "0." As shown in Figure 5 (page 63), the clock uses a total of 24 addresses rather than the whole block of 32 allocated to it. This permits additional design flexibility if port addresses are at a premium.

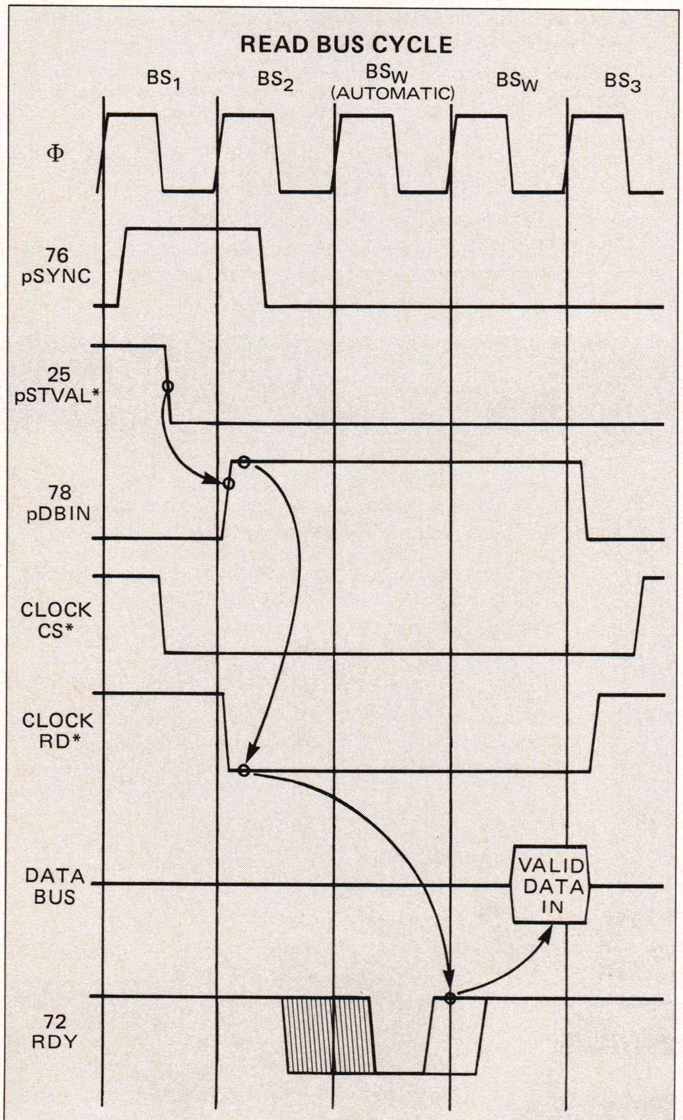


Figure 4.
Read Bus Cycle

You may install the board in any convenient location on the bus. Because of its wire-wrap construction, however, the board takes up the space of two normal boards and cannot be crowded. Depending on your needs, you can snip down the wire-wrap pins considerably (watch your eyes) to make the board thinner.

Software

When writing the programs to set the clock and print the time, I concentrated on writing clear, structured, modular code. Thus, you should have little difficulty making changes to transport the clock software to another system or to make modifications. Naturally, the resulting code perhaps is not as brief as might have been possible were I writing only for myself.

I have presented several programs here for using the clock. The first is CLOCKSET (see Listing One, page 68), a program to set either the day and date, the time, or both. The second is TIME (Listing Two, page 74), a program to display the time and date on the console. This same program will print the time and date plus a short text (such as a filename) to LST: if you include the text with the program invocation. This second feature of TIME is handy when you are using a text editor and need a date on a page about to be printed. The last is TIMELIST (see Listing Three, page 79), a program to print a time/date header with filename then list a text file with 60 lines per page.

Recall that I chose to use A0h as a clock base address for I/O. In all the programs, I used CLOCK as the base address and set its value equal to A0h. I identify other clock port addresses by a simple reference to CLOCK+1, CLOCK+2, and so on. For example, from Figure 5, CLOCK+3 is the address for "minutes" and CLOCK+7 is the address for "months." Using CLOCK in this way allows you to make an easy change to any base address just by changing one statement at the beginning of the program.

The first program, CLOCKSET, asks whether or not to set the date. If you select the default "No," then it asks whether to set the time. If you choose a second default "No," the program terminates without changing the clock. If, however, you do select the date-set option, then it asks for two digits for the day of the week: enter 01 for Sunday, 02 for Monday, and so on. You enter the day of the month the same way: 01 through 31. You enter the month itself as month 01 through 12. Select the year by entering just the last two digits; that is, 85 for 1985. Each January, you must change the year manually; the clock IC does not automatically change to a new year. When writing this program, I opted to keep it simple: no error checks and nothing fancy like entering literal names instead of numbers. If I make an error, I just invoke the program over again and do it correctly the next time through.

The second part of CLOCKSET is similar to the date-setting procedure. You enter the time in two-digit groups for the hours and minutes, and a carriage return sets the

Functional Description

TABLE II. ADDRESS CODES AND FUNCTIONS

A4	A3	A2	A1	A0	Function
0	0	0	0	0	Counter—Ten Thousandths of Seconds
0	0	0	0	1	Counter—Hundredths and Tenths of Seconds
0	0	0	1	0	Counter—Seconds
0	0	0	1	1	Counter—Minutes
0	0	1	0	0	Counter—Hours
0	0	1	0	1	Counter—Day of Week
0	0	1	1	0	Counter—Day of Month
0	0	1	1	1	Counter—Month
0	1	0	0	0	RAM—Ten Thousandths of Seconds
0	1	0	0	1	RAM—Hundredths and Tenths of Seconds
0	1	0	1	0	RAM—Seconds
0	1	0	1	1	RAM—Minutes
0	1	1	0	0	RAM—Hours
0	1	1	0	1	RAM—Day of Week
0	1	1	1	0	RAM—Day of Month
0	1	1	1	1	RAM—Months
1	0	0	0	0	Interrupt Status Register
1	0	0	0	1	Interrupt Control Register
1	0	0	1	0	Counters Reset
1	0	0	1	1	RAM Reset
1	0	1	0	0	Status Bit
1	0	1	0	1	GO Command
1	0	1	1	0	$\overline{\text{STANDBY INTERRUPT}}$
1	1	1	1	1	Test Mode

Figure 5.
Clock Address Codes (material courtesy of National Semiconductor Corporation)

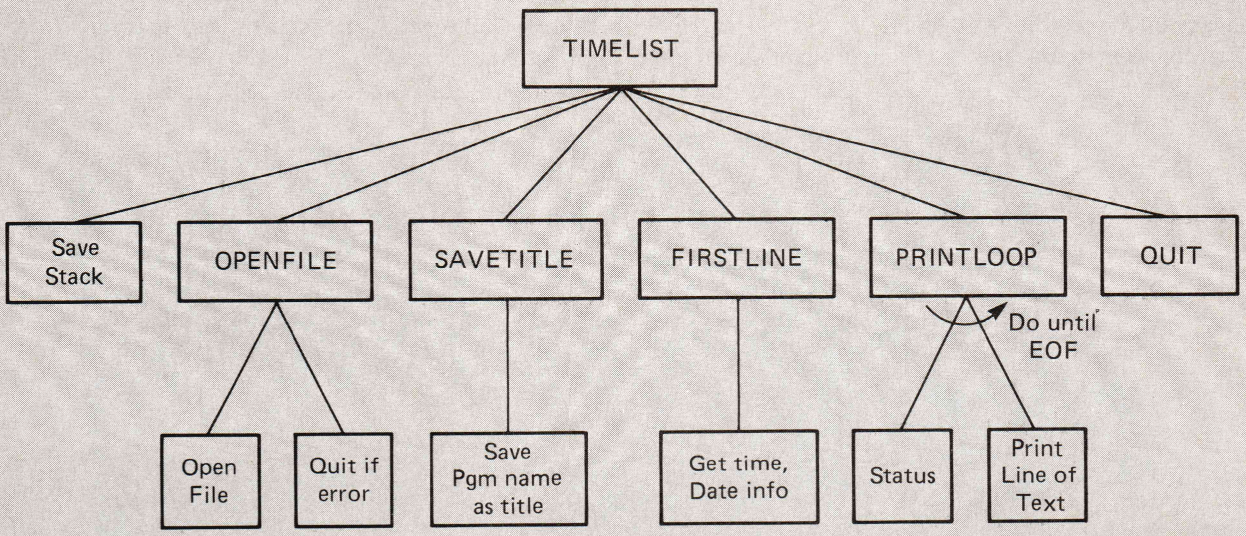


Figure 6
Structure Chart of the TIMELIST Program

seconds to zero. The clock chip keeps its time based on a 24-hour count, so if the time is 7:15 pm, for example, then the entry is 19 hours, 15 minutes. The carriage return to set seconds to zero does a write-data to CLOCK+21 (the GO command, see Figure 5). This write pulse resets the thousandths, hundredths, tenths, units, and tens of seconds counters. In the SETTIM subroutine, note that I reset the seconds (an early GO) before setting the minutes; this is because the clock chip will increment the minute counter upon GO if the seconds counter is greater than 40. For example, if the time is almost 7:15 pm and I enter 19 and 15 followed by <cr> to zero seconds, I might find the time erroneously set to 19:16; hence, the need to reset GO.

In the program TIME, unlike the simpler CLOCKSET program, I wanted more than just bare numbers from the clock: I wanted the literal names for days and months. The subroutine NTOLIT makes the conversion from the clock's packed BCD to a literal string. It does this by converting the number (say, 11 for November) to binary then indexing through the list of literals until it reaches the eleventh one. Then it moves the word "November" in memory to the space reserved for the message to be printed.

You can use TIME in two ways: either as a simple display to the console or as a display plus an output to the printer. To get the display, type TIME. To get the printer output in addition to the console display, type TIME <text>. The additional letters will send the time and date information plus the <text> to the currently defined LST: device connected to the computer. This is one way of time/date-stamping a page before sending some special output to the printer. Because the print output is aligned toward the right-hand side of the page, you have space for only about 15 characters before the printer wraps around. I use the space to put in a filename or some other text identification. Another way I use this feature is to print a disk name as <text> then send a disk directory to the printer; I keep this directory listing with the disk so I have a dated record of its contents.

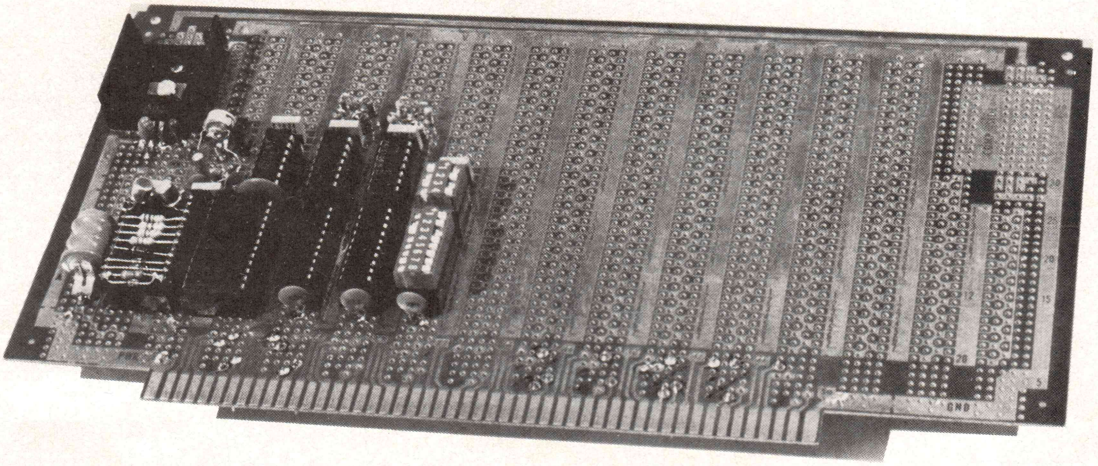
For convenient time/date-stamping of a full program listing, I chose the approach in the structure chart shown in Figure 6 (page 63). This program is invoked by TIMELIST <filename>. First, the title line is printed in the format: time, date, program name. Then, after skipping several lines, the printout of the text of the program begins, with a total of 60 lines on each page. All of the printing is done in subroutine PRINTLOOP, and you can interrupt it anytime from the keyboard; that is, a control-S pauses and any keypress continues, unless the printing is not paused, in which case any keypress aborts the program.

The clock-related aspects of this program are similar to TIME. The obvious difference is that a file must be opened, read, and printed. At the very beginning, then, because I used the 128 bytes between 80H and 100H for a disk buffer area, it is necessary to move the stack to a safe place. Then the first subroutine, OPENFILE, either opens the file successfully or calls QUIT if it encounters a problem. On exit, the stack pointer is restored to its previous position.

PRINTLOOP sends one character for each main loop within the subroutine. Before getting each new character, the program checks to see if a console break is present, to verify required spaces at the beginning of a line, and to determine if the maximum number of lines per page has been reached. The program as now assembled adds three spaces at the left margin for all printed text; this is because I usually notebook-punch my programs and my Okidata 82A prints too closely to the left-hand margin. To change this margin setting, adjust the SKIP equate. Likewise, to change from 60 lines per page, modify the LPAGE equate.

After getting a new byte by a call to GETNB, I check for the form-feed character and ignore it. To allow form feeds from the text, change the KEEPFF equate to TRUE. When accepting form feeds from the text, I disabled the line-counting within PRINTLOOP on the assumption that page length would have already been determined.

Before actually printing the character, the program checks for the proper line length. This is necessary because I skip several spaces at the left margin. If the text runs to 80 characters, but I've already printed three



The real-time clock on a Vector 8800 V wire-wrap board.

spaces, then three characters will have to wrap around to another line on the printer. To maintain the left margin, I need to insert the usual spaces before beginning to print the wrap-around letters.

If the character to be printed is a tab character (09), then TABTOSP does a conversion to spaces so that the printer lines up to every eighth column. The conversion is *not* simply to replace every tab with eight spaces: the program checks to see what the current column position is in relation to every eighth column then inserts just enough spaces to get the next eighth position. Thus, a tab positions the printer to columns 9, 17, 25, 33, and so on. Change the NRSPCS equate to the desired tab conversion if eight is not suitable.

As with all programs, a number of modifications always seem necessary. The CLOCKSET, TIME, and TIMELIST programs are certainly not the final versions. Improvements are always possible; it becomes an issue of whether the programs are "good enough to do the job" or *must* be revised. I chose to leave these three programs as they stand and go on to other more creative programs.

One program is to display the current time on the console, but I have a wristwatch that displays time quite well. Consider a "reminder" program to sound the console bell at a certain time. Also consider how to keep tax records of business use of the computer: a program to LOGON and LOGOFF that records date, time on/off, and subject of computer use. This program should create a file suitable for use by a spreadsheet program for tax records. These programs and others using the interrupt features of the clock are the subject of a future article.

Operation

When first starting up the clock, set the proper time and date using CLOCKSET. Then run the TIME program to see that the time and date are being displayed properly. Also check the printing feature of TIME by appending a small text to the TIME invocation to see that the time and text go to the printer. Test run TIMELIST with a two- or three-page text file to see that it prints the first page with time and date then prints 60 lines on each of the following pages.

Conclusions

The clock board has proven to be a useful addition to my computer system. I've found it especially convenient for dating disk directory prints, as well as for the normal dating of program printouts. Keeping the time and date on my program listings has certainly improved my program filing system: now when I discover a long-lost paper, I know when it disappeared!

I would like to thank our Bucknell Technician, Thomas J. Thul, Jr., for his help in laying out and constructing the clock circuit board. Program source code is available from the author on an 8-inch CP/M disk for $15 postpaid.

References

[1] Calaway, J. L., and B. Hill, "CP/M, Your Time Has Come," *BYTE*, May 1982, pp. 479-493.

[2] Hassebrock, Glen E., Jr., "The Hayes Chronograph, an H-89, and CP/M," *REMark*, No. 35, December 1982, pp. 9-14.

[3] Ciarcia, Steve, "Everyone Can Know the Real Time," *BYTE*, May 1982, pp. 34-58.

[4] *MM58167A Microprocessor Real Time Clock*, National Semiconductor Data Sheet, April 1982.

[5] *IEEE Standard 696 Interface Devices*, New York, NY: IEEE, 1983.

[6] Libes, Sol, and Mark Garetz, *Interfacing to S-100/IEEE-696 Microcomputers*, Berkeley, CA: Osborne/McGraw-Hill, 1981.

[7] *The TTL Data Book for Design Engineers*, 2nd ed., Dallas, TX: Texas Instruments, Inc., 1981.

Trademarks

Hardware:	IEEE Standard 696-1983 compatible
	Operation to 6 MHz (using two wait states)
	Handshaking using RDY line (72) pulled low until clock output data valid
	DIP-switch selection of base address on any 32-byte I/O port boundary, 32 ports required
	DIP-switch selection of vectored interrupt 0 through 7 for clock interrupts 100 ms to 1 month or RAM and counter compares
	DIP-switch selection of vectored interrupt 0 through 7 for RAM and counter compares
	Accuracy 0.01%, but generally better than 5 sec/month
	Power requirement +8V < 200 mA during normal operation
	Rechargeable battery backup
	Construction using Vector 8800V and wire-wrap
Software:	24-hour time: hour, minute, second
	Calendar: day of week and month, month, year
	File-list utility with time/date header
	Clock time- and date-setting program

Table
Specifications of the S-100 Real-Time Clock

DDJ

(Listing begins on page 68)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 194.**

Listing One

```
                        ; CLOCKSET      Program to set date & time of real-time clock
                ;
                ; Program by     Alan D. Wilcox
                ;                6 January 1985
                ;
0100                    ORG     100H

00A0 =          CLOCK   EQU     0A0H     ; Clock base address

0000 =          FALSE   EQU     0
00FF =          TRUE    EQU     0FFH

0005 =          BDOS    EQU     5H       ; BDOS entry point

0001 =          CONSIN  EQU     1        ; Console input
0002 =          CONSOUT EQU     2        ; Console output
0009 =          PRINTST EQU     9        ; Print string function
000B =          GETSTAT EQU     11       ; Console status

000A =          LF      EQU     0AH      ; Line Feed
000D =          CR      EQU     0DH      ; Carriage Return
001B =          ESC     EQU     1BH      ; Escape
                ;
                ;
                ; ***** MAIN PROGRAM *****
                ; *
                ; *

0100 113202     START:  LXI     D,GREET
0103 CD1302             CALL    PRINTIT

0106 116502             LXI     D,DATESET        ; Set the date?
0109 CD1302             CALL    PRINTIT
010C CD1B02             CALL    GETANS           ; Get answer
010F FEFF               CPI     TRUE
0111 CC2901             CZ      SETDATE

0114 117F02             LXI     D,TIMESET        ; Set the time?
0117 CD1302             CALL    PRINTIT
011A CD1B02             CALL    GETANS           ; Get answer
011D FEFF               CPI     TRUE
011F CC6401             CZ      SETTIME

0122 119F03             LXI     D,CRLF
0125 CD1302             CALL    PRINTIT

0128 C9         DONE:   RET                      ; Return to CP/M

                ; *
                ; *
                ;
                ; ***** SUBR SETDATE *****
                ;
                ;       Set the current date
                ;
0129 119902     SETDATE:LXI     D,DAY            ; What day of week?
012C CD1302             CALL    PRINTIT
012F CDDB01             CALL    GETBCD           ; Get it and store
0132 D3A5               OUT     CLOCK+5          ;  in clk counter

0134 11C402             LXI     D,DATE           ; What date is it?
0137 CD1302             CALL    PRINTIT
013A CDDB01             CALL    GETBCD           ; Get it and store
013D D3A6               OUT     CLOCK+6          ;  in clk counter

013F 11E002             LXI     D,MONTH          ; What month is it?
0142 CD1302             CALL    PRINTIT
0145 CDDB01             CALL    GETBCD           ; Get it and store
0148 D3A7               OUT     CLOCK+7          ;  in clk counter

014A 11FC02             LXI     D,YEAR           ; And the year?
014D CD1302             CALL    PRINTIT
```

(Continued on page 70)

Listing One

```
0150 CDDB01              CALL    GETBCD          ; Get it and store
0153 D3A9                OUT     CLOCK+9         ;  in clock RAM

0155 118A03              LXI     D,AGAIN         ; Do this over again?
0158 CD1302              CALL    PRINTIT
015B CD1B02              CALL    GETANS          ; Get answer
015E FEFF                CPI     TRUE
0160 CA2901              JZ      SETDATE         ; Yes, do it.

0163 C9                  RET
                ;
                ;
                ; ***** SUBR SETTIME *****
                ;
                ;        Set the current time
                ;
0164 CD9C01     SETTIME:CALL     DISPLAY         ; Show the time at start

0167 113503              LXI     D,SETHR         ; Set-hrs message
016A CD1302              CALL    PRINTIT
016D CDDB01              CALL    GETBCD          ; Get 2 ASCII, convert to BCD
0170 D3A4                OUT     CLOCK+4         ; Put hours into clock cnter

0172 115203              LXI     D,SETMIN        ; Set-min message
0175 CD1302              CALL    PRINTIT
0178 D3B5                OUT     CLOCK+21        ; Reset seconds
017A CDDB01              CALL    GETBCD          ; Get 2 ASCII, convert to BCD
017D D3A3                OUT     CLOCK+3         ; Put mins into clock counter

017F 116E03              LXI     D,SETSEC        ; Set-seconds message
0182 CD1302              CALL    PRINTIT
0185 CD0B02              CALL    INCHAR          ; Get key-press
0188 D3B5                OUT     CLOCK+21        ; "GO" command zero seconds

018A CD9C01              CALL    DISPLAY         ; Show the results

018D 118A03              LXI     D,AGAIN         ; Ask to do again
0190 CD1302              CALL    PRINTIT
0193 CD1B02              CALL    GETANS
0196 FEFF                CPI     TRUE
0198 CA6401              JZ      SETTIME         ; Yes, do it again

019B C9                  RET
                ;
                ;
                ; ***** SUBR DISPLAY *****
                ;
                ;        Displays current time
                ;
019C 111803     DISPLAY:LXI      D,TMSG          ; Time is ...
019F CD1302              CALL    PRINTIT

01A2 DBA4                IN      CLOCK+4         ; Get hours from clock
01A4 CDBC01              CALL    PRHEX           ; Print hours
01A7 3E3A                MVI     A,':'
01A9 CD0402              CALL    PCHAR

01AC DBA3                IN      CLOCK+3         ; Get minutes from clock
01AE CDBC01              CALL    PRHEX           ; Print minutes
01B1 3E3A                MVI     A,':'
01B3 CD0402              CALL    PCHAR

01B6 DBA2                IN      CLOCK+2         ; Get seconds
01B8 CDBC01              CALL    PRHEX           ; Print seconds

01BB C9                  RET
                ;
                ;
                ; ***** SUBR PRHEX *****
                ;
                ;        Converts binary value into two ASCII-hex characters
                ;        and prints on console
```

(Continued on page 72)

Listing One

```
                        ;
01BC F5          PRHEX:  PUSH    PSW
01BD OF                  RRC                          ; Put 4 MSB's into 4 LSB's
01BE OF                  RRC
01BF OF                  RRC
01C0 OF                  RRC
01C1 CDC901              CALL    PRNIB                ; Prnt hex equiv to 4 MSB's
01C4 F1                  POP     PSW
01C5 CDC901              CALL    PRNIB                ; Prnt hex equiv to 4 LSB's
01C8 C9                  RET
                 ;
                 ;
                 ;  ***** SUBR PRNIB *****
                 ;
                 ;       Prints a nibble of Reg A
                 ;
01C9 E60F        PRNIB:  ANI     OFH                  ; Mask out top 4 bits
01CB FE0A                CPI     10                   ; Is it number or letter?
01CD D2D501              JNC     LETR                 ; Must be a letter if jump.
01D0 C630                ADI     '0'                  ; Add offset to make  ASCII
01D2 C3D701              JMP     PRNT
01D5 C637        LETR:   ADI     'A' - 10             ; Add offset to make binary
                                                      ;  into ASCII letter
01D7 CD0402      PRNT:   CALL    PCHAR                ; Send ASCII to console
01DA C9                  RET
                 ;
                 ;
                 ;  ***** SUBR GETBCD *****
                 ;
                 ;       Gets 2 ASCII characters, converts them to packed BCD
                 ;
                 ;       EXIT:   A = 2 BCD characters
                 ;
01DB CD0B02      GETBCD: CALL    INCHAR               ; Get character
01DE FE30                CPI     '0'
01E0 DADB01              JC      GETBCD               ; Ignore char if < ASCII 0
01E3 FE3A                CPI     '9'+1
01E5 D2DB01              JNC     GETBCD               ; Ignore char if > ACCII 9
01E8 D630                SUI     '0'                  ; Make ASCII into binary
01EA E60F                ANI     OFH                  ; Mask out 4 MSB's
01EC 07                  RLC
01ED 07                  RLC
01EE 07                  RLC
01EF 07                  RLC                          ; Put into MSB position
01F0 47                  MOV     B,A                  ;  and save in B
01F1 CD0B02      LOBYTE: CALL    INCHAR               ; Get 2nd character
01F4 FE30                CPI     '0'
01F6 DAF101              JC      LOBYTE
01F9 FE3A                CPI     '9'+1
01FB D2F101              JNC     LOBYTE
01FE D630                SUI     '0'                  ; Make ASCII into binary.
0200 E60F                ANI     OFH
0202 80                  ADD     B                    ; Combine both BCD char in A.
0203 C9                  RET
                 ;
                 ;
                 ;  ***** SUBR PCHAR *****
                 ;
                 ;       Prints Reg A to console
                 ;
0204 0E02        PCHAR:  MVI     C,CONSOUT
0206 5F                  MOV     E,A
0207 CD0500              CALL    BDOS
020A C9                  RET
                 ;
                 ;
                 ;  ***** SUBR INCHAR *****
                 ;
                 ;       Reg A gets ASCII value from keyboard
                 ;
020B C5          INCHAR: PUSH    B                    ; Have char saved in B
020C 0E01                MVI     C,CONSIN
```

(Continued on page 74)

Listing One

```
020E CD0500             CALL    BDOS
0211 C1                 POP     B
0212 C9                 RET
                ;
                ;
                ; ***** SUBR PRINTIT *****
                ;
                ;       ENTRY:  DE = Start address of string
                ;
0213 F5         PRINTIT:PUSH    PSW
0214 0E09               MVI     C,PRINTST
0216 CD0500             CALL    BDOS            ; Print string to console
0219 F1                 POP     PSW
021A C9                 RET
                ;
                ;
                ; ***** SUBR GETANS *****
                ;
                ;       Ask question, then call this subroutine for answer.
                ;       Assume default answer is NO.
                ;
                ;       EXIT:   A = True = FF if YES
                ;               A = False= 0  if NO
                ;
021B 0E01       GETANS: MVI     C,CONSIN
021D CD0500             CALL    BDOS            ; Get console input
0220 FE59               CPI     'Y'
0222 CA2F02             JZ      ENDTRU
0225 FE79               CPI     'y'
0227 CA2F02             JZ      ENDTRU
022A 3E00               MVI     A,FALSE         ; Not Y or y. Assume NO.
022C C33102             JMP     GOTANS
022F 3EFF       ENDTRU: MVI     A,TRUE
0231 C9         GOTANS: RET
                ;
                ;
                ; ***** CONSOLE MESSAGES *****
                ;
0232 0D0A50726F  GREET:    DB      CR,LF,'Program to set DATE & TIME of '
0252 7265616C2D            DB      'real-time clock.',LF,CR,'$'

0265 0D0A0A5365  DATESET:  DB      CR,LF,LF,'Set the date? <N> --> $'
027F 0D0A0A5365  TIMESET:  DB      CR,LF,LF,'Set the time? <N> --> $'

0299 0D0A0A5375  DAY:      DB      CR,LF,LF,'Sunday = 01.'
02A8 0D0A456E74            DB      CR,LF,'Enter day 01 to 07.  --> $'
02C4 0D0A456E74  DATE:     DB      CR,LF,'Enter date 01 to 31  --> $'
02E0 0D0A456E74  MONTH:    DB      CR,LF,'Enter month 01 to 12 --> $'
02FC 0D0A456E74  YEAR:     DB      CR,LF,'Enter year 84 to 99  --> $'

0318 0D0A0A4872  TMSG:     DB      CR,LF,LF,'Hrs : Min : Sec now   --> $'
0335 0D0A0A496E  SETHR:    DB      CR,LF,LF,'Input HRs 00 to 23   --> $'
0352 0D0A496E70  SETMIN:   DB      CR,LF,'Input Min 00 to 59   --> $'
036E 0D0A507265  SETSEC:   DB      CR,LF,'Press <cr> to zero sec...$'

038A 0D0A0A446F  AGAIN:    DB      CR,LF,LF,'Do over? <N> ... $'

039F 0D0A24      CRLF:     DB      CR,LF,'$'

03A2                    END
```

End Listing One

Listing Two

```
        ; TIME          Display time and date of real-time clock
        ;
        ; Program by    Alan D. Wilcox
        ;               18 December 1984
        ;
        ; Invoke by     A>TIME          Output to Console
        ;               A>TIME <text>   Output to Console & LST:
```

```
                     ;                                (List time & text)
 0100                          ORG     100H

 00A0 =              CLOCK     EQU     0A0H      ; Clock base address

 0005 =              BDOS      EQU     5H        ; BDOS entry point
 0080 =              BUFF      EQU     80H       ; Buffer

 0005 =              LISTOUT   EQU     5         ; List output
 0009 =              PRINTST   EQU     9         ; Print string function

 000A =              LF        EQU     0AH       ; Line Feed
 000D =              CR        EQU     0DH       ; Carriage Return
 001B =              ESC       EQU     1BH       ; Escape
                     ;
                     ;
                     ; ***** MAIN PROGRAM *****
                     ; *
                     ; *
                     ;         Read current time and save in memory

0100 DBA4       GETIME: IN      CLOCK+4           ; Get hours from clock cntr
0102 11D701             LXI     D,HOURS
0105 CD6D01             CALL    BCDTOASC          ; Convert to ASCII & save

0108 DBA3               IN      CLOCK+3           ; Get minutes from clk cntr
010A 11DA01             LXI     D,MINUTES
010D CD6D01             CALL    BCDTOASC          ; Convert

0110 DBA2               IN      CLOCK+2           ; Get seconds
0112 11DD01             LXI     D,SECONDS
0115 CD6D01             CALL    BCDTOASC

                ;       Read day of week and save literal equiv. in memory

0118 DBA5       GETDAY: IN      CLOCK+5
011A 11E201             LXI     D,DAY             ; Where to save the result
011D 211302             LXI     H,LITDAY          ; Location of literal weekday
0120 CD7F01             CALL    NTOLIT            ; Convert number to literal

                ;       Read date and save in memory

0123 DBA6       GETDATE:IN      CLOCK+6
0125 11EE01             LXI     D,DATE            ; Where to save result
0128 CD6D01             CALL    BCDTOASC          ; Convert to ASCII and save

                ;       Read month and save literal equivalent in memory

012B DBA7       GETMON: IN      CLOCK+7
012D 11F101             LXI     D,MONTH           ; Where to save result
0130 215902             LXI     H,LITMON          ; Location of literal months
0133 CD7F01             CALL    NTOLIT            ; Convert to literal

                ;       Read year from clock RAM & save in memory

0136 DBA9       GETYR:  IN      CLOCK+9
0138 11FD01             LXI     D,YEAR            ; Where to save result
013B CD6D01             CALL    BCDTOASC          ; Convert to ASCII and save

                ;       Send time message to console

013E 11CE01     CONSL:  LXI     D,MSGL            ; DE gets adr of message
0141 0E09               MVI     C,PRINTST
0143 CD0500             CALL    BDOS              ; Print string to console

                ;       Send time msg to printer if cmd string required it

0146 3A8000     PRINT:  LDA     BUFF              ; Was cmd 'TIME <text>'
0149 FE00               CPI     0                 ; Done if nothing there
014B CA6C01             JZ      DONE

014E 4F                 MOV     C,A               ; Put buffer count in C
014F 218100             LXI     H,BUFF+1          ; HL is source of text
0152 110102             LXI     D,TEXT            ; DE is destin for text

0155 7E         MOVETXT:MOV     A,M               ; Get byte
0156 12                 STAX    D                 ; Save it
```

(Continued on next page)

Listing Two

```
0157 23                 INX     H
0158 13                 INX     D
0159 0D                 DCR     C
015A C25501             JNZ     MOVETXT          ; Do until C is 0

015D 3E24               MVI     A,'$'            ; End-text marker
015F 12                 STAX    D

0160 21BD01             LXI     H,MSGR           ; Go ahead to printer
0163 CDA701             CALL    LISTIT
0166 21BA01             LXI     H,CRLF           ; Finish the line
0169 CDA701             CALL    LISTIT

016C C9         DONE:   RET                      ; Return to CP/M

                ; *
                ; *

                ; ***** SUBR BCDTOASC *****
                ;
                ;       Converts binary value into two ASCII-hex characters
                ;
                ;       ENTRY:  Reg A  = 2 packed-BCD characters
                ;                   DE = Adr of where to store ASCII answers
                ;
                ;       EXIT:     (DE) = MS char
                ;               (DE+1) = LS char
                ;
                BCDTOASC:
016D F5                 PUSH    PSW
016E 0F                 RRC                      ; Put 4 MSB's into 4 LSB's
016F 0F                 RRC
0170 0F                 RRC
0171 0F                 RRC
0172 E60F               ANI     0FH              ; Mask out top 4 bits
0174 C630               ADI     '0'              ; Add offset to make  ASCII
0176 12                 STAX    D                ; Save the result
0177 F1                 POP     PSW              ; Get the next character
0178 E60F               ANI     0FH
017A C630               ADI     '0'              ; Make ASCII
017C 13                 INX     D
017D 12                 STAX    D                ; Save it in next memory loc.
017E C9                 RET


                ; ***** SUBR NTOLIT *****
                ;
                ;       Converts packed-BCD number (1 to 12) to its equiv
                ;       literal string and saves in specified memory.
                ;
                ;       ENTRY:  A  =  2 packed-BCD numbers from clock
                ;               DE =  Adr of where to store literal results
                ;               HL =  Pntr to 10-char literal strings
                ;
017F 47         NTOLIT: MOV     B,A              ; Save data
0180 E610               ANI     00010000B        ; See if have tens digit
0182 CA8B01             JZ      SMALLNR          ; Jump if not
0185 78                 MOV     A,B
0186 C60A               ADI     10               ; Add 10 to low nibble so
0188 C38C01             JMP     BINARY           ;  it's binary now.
018B 78         SMALLNR:MOV     A,B

018C E60F       BINARY: ANI     0FH              ; Mask off top nibble
018E D601               SUI     1                ; Make number 0 to 11 max
0190 CA9B01             JZ      MOVE             ; At first already

0193 010A00             LXI     B,10             ; 10 literals in word
0196 09         NEXT:   DAD     B                ; Add it to HL for next word
0197 3D                 DCR     A                ;  until get to proper one.
0198 C29601             JNZ     NEXT
```

(Continued on page 78)

Listing Two

```
019B 010A00    MOVE:    LXI     B,10           ; Move 10 letters only

019E 7E        MOVDAT:  MOV     A,M            ; HL pts to desired word
019F 12                 STAX    D              ; DE pts to destination mem.
01A0 13                 INX     D
01A1 23                 INX     H
01A2 0D                 DCR     C              ; Proper literal word now
01A3 C29E01             JNZ     MOVDAT         ;  moved to memory to print

01A6 C9                 RET

                ; ***** SUBR LISTIT *****
                ;
                ;       Lists to printer until '$' encountered.
                ;
                ;       ENTRY:  HL  =  string address
                ;
01A7 7E        LISTIT:  MOV     A,M            ; Get the letter to be sent
01A8 FE24               CPI     '$'            ; End of text yet?
01AA CAB901             JZ      LISTEND
01AD E5                 PUSH    H              ; Must retain HL!
01AE 5F                 MOV     E,A
01AF 0E05               MVI     C,LISTOUT
01B1 CD0500             CALL    BDOS           ; Print the char
01B4 E1                 POP     H
01B5 23                 INX     H
01B6 C3A701             JMP     LISTIT         ; Keep going until '$'
01B9 C9        LISTEND:RET

                ; ***** MESSAGE AREA *****
                ;
01BA 0D0A24      CRLF:      DB      CR,LF,'$'

01BD 2020202020  MSGR:      DB      '                  '
01CC 2020                   DB      '  '
01CE 2020202020  MSGL:      DB      '          '
01D7             HOURS:     DS      2
01D9 3A                     DB      ':'
01DA             MINUTES:   DS      2
01DC 3A                     DB      ':'
01DD             SECONDS:   DS      2
01DF 202020                 DB      '   '
01E2             DAY:       DS      10
01EC 2020                   DB      '  '
01EE             DATE:      DS      2
01F0 20                     DB      ' '
01F1             MONTH:     DS      10
01FB 3139                   DB      '19'
01FD             YEAR:      DS      2
01FF 2020                   DB      '  '
0201 2020202020  TEXT:      DB      '               '
0210 0D0A24                 DB      CR,LF,'$'

                ; ***** LITERALS *****
                ;
                ;        Pack with NULLs for even spacing on print line
                ;
0213 53756E6461 LITDAY:    DB      'Sunday,',0,0,0
021D 4D6F6E6461            DB      'Monday,',0,0,0
0227 5475657364            DB      'Tuesday,', 0,0
0231 5765646E65            DB      'Wednesday,'
023B 5468757273            DB      'Thursday,',  0
0245 4672696461            DB      'Friday,',0,0,0
024F 5361747572            DB      'Saturday,',  0

0259 4A616E7561 LITMON:    DB      'January ',    0,0
0263 4665627275            DB      'February ',     0
026D 4D61726368            DB      'March ',  0,0,0,0
```

```
0277 417072696C            DB      'April ',  0,0,0,0
0281 4D61792000            DB      'May ',0,0,0,0,0,0
028B 4A756E6520            DB      'June ', 0,0,0,0,0
0295 4A756C7920            DB      'July ', 0,0,0,0,0
029F 4175677573            DB      'August ',    0,0,0
02A9 5365707465            DB      'September '
02B3 4F63746F62            DB      'October ',     0,0
02BD 4E6F76656D            DB      'November ',      0
02C7 446563656D            DB      'December ',      0

02D1                 END
```

End Listing Two

Listing Three

```
                    ; TIMELIST       Program to read a file and send it out
                    ;                to printer with time/date and program name
                    ;
                    ; Program by     Alan D. Wilcox
                    ;                29 October 1984
                    ;
                    ; Usage          A> TIMELIST  filespec.txt
                    ;

 0100                      ORG     100H

 00A0 =             CLOCK  EQU     0A0H    ; Clock base address

 0000 =             FALSE  EQU     0
 FFFF =             TRUE   EQU     NOT FALSE

 FFFF =             TOLPTR EQU     TRUE    ; Make FALSE to send all output to
                                           ;  console rather than to line prntr

 0000 =             KEEPFF  EQU    FALSE   ; Make TRUE to use form feeds in
                                           ;  orig text. TRUE also cancels line
                                           ;  counting by this program.

 0005 =             BDOS    EQU    0005H   ; BDOS entry point

 0002 =             CONSOUT EQU    2       ; Console Output
 0005 =             LISTOUT EQU    5       ; List Output
 0009 =             PRINTST EQU    9       ; Print String Function
 000B =             GETSTAT EQU    11      ; Console Status
 000F =             OPENF   EQU    15      ; Open File
 0014 =             READF   EQU    20      ; Read Next Record

 005C =             FCB     EQU    5CH     ; File Control Block Address
 007C =             FCBCR   EQU    FCB+32  ; Current (next) record for r/w
 0080 =             BUFF    EQU    80H     ; Input Disk Buffer Address

 000C =             TSIZE   EQU    12      ; Title block size
 0003 =             SKIP    EQU    3       ; Added number of spcs at left margn
 0050 =             COLS    EQU    80      ; Max possible columns per page
 003C =             LPAGE   EQU    60      ; Lines to print per page
 0008 =             NRSPCS  EQU    8       ; Convert tab to NRSPCS spaces

 0009 =             TAB     EQU    09H     ; Horizontal Tab
 000A =             LF      EQU    0AH     ; Line Feed
 000B =             VT      EQU    0BH     ; Vertical Tab (alt FF)
 000C =             FF      EQU    0CH     ; Form Feed
 000D =             CR      EQU    0DH     ; Carriage Return
 001A =             CNTLZ   EQU    1AH     ; End of File (cntl-z)
 001B =             ESC     EQU    1BH     ; Escape
 0020 =             SPC     EQU    20H     ; Space
                    ;
                    ;
                    ; ********** MAIN PROGRAM **********
                    ; *
                    ; *

 0100 210000        STACK:  LXI     H,0             ; Stack in safe place while
 0103 39                    DAD     SP              ;   using disk buffer space.
 0104 227904                SHLD    OLDSTK          ; Save old stack position
```

(Continued on next page)

Listing Three

```
0107 319F04            LXI     SP,NEWSTK          ; Use new stack pointer

010A CD1B01            CALL    OPENFILE           ; Open program to be read

010D CD3B01            CALL    SAVETITLE          ; Save pgm name as the title

0110 CD7901            CALL    FIRSTLINE          ; Prnt time/date/title line

0113 CDD301            CALL    PRINTLOOP          ; Read & print file

0116 2A7904    QUIT:   LHLD    OLDSTK             ; Get the old stack pntr in
0119 F9                SPHL                       ;  HL & put in on the stack.
011A C9                RET                        ; Return to CP/M

               ; *
               ; *
               ;
               ; **********    SUBR OPENFILE     **********
               ;
               OPENFILE:       ; Open the file requested

011B AF                XRA     A                  ; Zero accumulator
011C 327C00            STA     FCBCR              ; Zero file record count
011F 115C00            LXI     D,FCB
0122 0E0F              MVI     C,OPENF            ; Open the file
0124 CD0500            CALL    BDOS               ; Get 0FFH in Accum if error
0127 FEFF              CPI     0FFH
0129 C23501            JNZ     OKOPEN

012C 113D03            LXI     D,OPENERR          ; Had error in file opening
012F CD3503            CALL    PRINTIT            ; Print error message
0132 CD1601            CALL    QUIT               ; Restore stack and get out

0135 3E80      OKOPEN: MVI     A,BUFF             ; Set position of input
0137 327B04            STA     INBUFPT            ;  buffer pointer
013A C9                RET
               ;
               ;
               ; **********    SUBR SAVETITLE    **********
               ;
               SAVETITLE:      ; Save name of program as title

013B 0E0C              MVI     C,TSIZE            ; Clear TSIZE space in memory
013D 21AA03            LXI     H,HEADER           ;  starting at HEADER.
0140 3620      CLRBLK: MVI     M,SPC              ; Fill with spaces
0142 23                INX     H
0143 0D                DCR     C
0144 C24001            JNZ     CLRBLK
0147 3A8300            LDA     BUFF+3             ; See if drive in cmd line
014A FE3A              CPI     ':'                ; Z set if true
014C C25A01            JNZ     NODRIV
014F 118400            LXI     D,BUFF+4           ; DE pts to pgm name only
0152 3A8000            LDA     BUFF               ; Get count of chars
0155 D603              SUI     3                  ; Sub cnt, space, drive spec
0157 C36201            JMP     ADJSIZE

015A 118200    NODRIV: LXI     D,BUFF+2           ; DE points to pgm name
015D 3A8000            LDA     BUFF               ; Get count of chars
0160 D601              SUI     1                  ; Subtr count & space

0162 FE0C      ADJSIZE:CPI     TSIZE              ; Adjust char cnt so it fits
0164 F26B01            JP      BIG                ; Jump if count > TSIZE
0167 4F                MOV     C,A                ; Acceptable count into C
0168 C36D01            JMP     OKSIZE
016B 0E0C      BIG:    MVI     C,TSIZE            ; Title is max size

016D 21AA03    OKSIZE: LXI     H,HEADER           ; Destination for title move

0170 1A        MTITLE: LDAX    D                  ; Get the character
0171 77                MOV     M,A                ;  and save it as header
0172 23                INX     H
0173 13                INX     D
```

```
0174 0D                 DCR     C
0175 C27001             JNZ     MTITLE          ; Title saved when zero
0178 C9                 RET
               ;
               ;
               ; **********    SUBR FIRSTLINE   **********
               ;
               FIRSTLINE:      ; Read current time and save in memory

0179 DBA4               IN      CLOCK+4         ; Get hours from clock
017B 116B03             LXI     D,HOURS
017E CDE102             CALL    BCDTOASC        ; Convert to ASCII & save

0181 DBA3               IN      CLOCK+3         ; Get minutes from clock
0183 116E03             LXI     D,MINUTES
0186 CDE102             CALL    BCDTOASC        ; Convert

0189 DBA2               IN      CLOCK+2         ; Get seconds
018B 117103             LXI     D,SECONDS
018E CDE102             CALL    BCDTOASC

               ;        Read day of week and save literal equiv. in memory

0191 DBA5               IN      CLOCK+5
0193 118003             LXI     D,DAY           ; Where to save the result
0196 21BB03             LXI     H,LITDAY        ; Location of literal weekday
0199 CDF302             CALL    NTOLIT          ; Convert number to literal

               ;        Read date and save in memory

019C DBA6               IN      CLOCK+6
019E 118B03             LXI     D,DATE          ; Where to save result
01A1 CDE102             CALL    BCDTOASC        ; Convert to ASCII and save

               ;        Read month and save literal equivalent in memory

01A4 DBA7               IN      CLOCK+7
01A6 118E03             LXI     D,MONTH         ; Where to save result
01A9 210104             LXI     H,LITMON        ; Location of literal months
01AC CDF302             CALL    NTOLIT          ; Convert to literal

               ;        Read year from clock RAM & save in memory

01AF DBA9               IN      CLOCK+9
01B1 119A03             LXI     D,YEAR
01B4 CDE102             CALL    BCDTOASC        ; Convert to ASCII and save

               ;        Clear spaces at left margin before printing time

01B7 0E03               MVI     C,SKIP
01B9 216803             LXI     H,TOPLINE
01BC 3620      CLRSKP:  MVI     M,SPC           ; Fill with spaces
01BE 23                 INX     H
01BF 0D                 DCR     C
01C0 C2BC01             JNZ     CLRSKP

               IF TOLPTR        ; Send time message to line printer
01C3 216803             LXI     H,TOPLINE       ; HL gets adr of message
01C6 CD1B03             CALL    LISTIT          ; To printer
               ENDIF
               ;
               IF NOT TOLPTR    ; Send time message to console
                        LXI     D,TOPLINE       ; DE gets adr of message
                        CALL    PRINTIT         ; Print to console
               ENDIF

               ;        Set line cntr to 3 to account for title block lines

01C9 3E03               MVI     A,3
01CB 327E04             STA     LINECNT

               ;        Initialize column counter

01CE AF                 XRA     A               ; Zero accum
01CF 327D04             STA     COLCNT          ; Column counter zero
01D2 C9                 RET
               ;
               ;
```

(Continued on next page)

Listing Three

```
                        ; **********    SUBR PRINTLOOP   **********
                        ;

                        PRINTLOOP:       ; Will stay in this loop until either break
                                         ;  from console or file is completely printed

                        ;        Check console for break. Exit without losing stack

01D3 0E0B                        MVI     C,GETSTAT
01D5 CD0500                      CALL    BDOS            ; Accum LSB=1 if brk request
01D8 0F                          RRC                     ; Put LSB into carry
01D9 DA5702                      JC      DONE            ; Leave program

                        ;        Check to see if need spaces at start of left margin

01DC 3A7D04                      LDA     COLCNT          ; Current column count
01DF FE03                        CPI     SKIP            ; Sign set while COLCNT< SKIP
01E1 F2ED01                      JP      CKLINE          ; Jump when COLCNT>=SKIP
01E4 3C                          INR     A               ; Increment col counter
01E5 327D04                      STA     COLCNT
01E8 3E20                        MVI     A,SPC
01EA C35102                      JMP     GOPRNT          ; Go print space & continue

                        ;        Check for proper lines per page. Send FF if needed

01ED 3A7E04             CKLINE:  LDA     LINECNT         ; Current line count
01F0 FE3C                        CPI     LPAGE           ; Max lines per page
01F2 C20102                      JNZ     DATA            ; Get data if not equal yet

                        IF KEEPFF
                                 JMP     DATA            ; Don't bother counting.
                        ENDIF                            ; Use FF's in source text.

01F5 AF                          XRA     A               ; Zero accum
01F6 327E04                      STA     LINECNT         ; Reset line cntr when equal
01F9 327D04                      STA     COLCNT          ; Reset column counter too
01FC 3E0C                        MVI     A,FF
01FE C35102                      JMP     GOPRNT          ; Print FF & continue

                        ;        Get the data and filter out various characters

0201 CD5B02             DATA:    CALL    GETNB           ; Get the next byte.
0204 DA5702                      JC      DONE            ; Carry set if EOF
0207 FE0D                        CPI     CR
0209 CA5102                      JZ      GOPRNT          ; Print the CR

020C FE0A               CKLF:    CPI     LF
020E C22102                      JNZ     CKFF            ; Jump if not a LF
0211 3A7E04                      LDA     LINECNT
0214 3C                          INR     A               ; Increment line counter
0215 327E04                      STA     LINECNT
0218 AF                          XRA     A
0219 327D04                      STA     COLCNT          ; Set col counter to zero
021C 3E0A                        MVI     A,LF            ; Restore data
021E C35102                      JMP     GOPRNT          ; Print the LF
                        IF KEEPFF
                        CKFF:    JMP     LNWID           ; Go and print FF & VT chars
                        ENDIF
                        ;
                        IF NOT KEEPFF
0221 FE0C               CKFF:    CPI     FF
0223 CA0102                      JZ      DATA            ; If FF, ignore. Get new data
0226 FE0B                        CPI     VT
0228 CA0102                      JZ      DATA            ; If VT, ignore it too.
                        ENDIF

                        ;        Check for proper width of line

022B F5                 LNWID:   PUSH    PSW
022C 3A7D04                      LDA     COLCNT          ; Get current column count
022F FE50                        CPI     COLS            ; Max allowed per line
0231 C24202                      JNZ     NOTMAX
```

(Continued on page 84)

GRAPHICS FROM YOUR DOT MATRIX PRINTER

HPLOT

A PLOTTER EMULATION PROGRAM FOR YOUR OKIDATA, PROWRITER, GEMINI, OR EPSON PRINTER.

* POWERFUL HP-GL PLOTTER SYNTAX: SCALING, LINETYPES, WINDOWS, ETC; LABELS ANY SIZE, *SLANT*, OR DIRECTION.
* FAST! GRAPHS IN FOUR MINUTES.
* HI-RES MODE: UP TO 136x144 DPI.
* PLOT SIZES 11"x14" TO 7"x48".
* 80+ PAGE ILLUSTRATED MANUAL.
* SOURCE CODE IN C FOR PROGRAMS THAT USE HPLOT TO MAKE PIE CHARTS, GRAPHS, ETC.
* REQUIRES 54K Z80 CP/M 2.2. OTHER PRINTERS AND OS'S SOON!
* AVAILABLE IN 8" SSSD AND MOST 5.25" 48 TPI FORMATS.

$49.95 PPD. OH RES ADD 5% TAX

ORDINATE SOLUTIONS
MAIN P.O. BOX 0308, OBERLIN, OH 44074

THIS AD WAS PREPARED ON AN OKIDATA 92.

Trademarks: CP/M (Digital Research), HP-GL (Hewlett-Packard), Z80 (Zilog)

TURBO ASSEMBLER

Introducing ***FAST ASSEM-86***™, the **TURBO PASCAL**™ of IBM PC assemblers. ***FAST ASSEM-86***™ ***(FASM)*** is significantly faster and easier to use than the IBM Macro-Assembler (MASM). Whether you are new to assembly language and want to quickly write a small assembly language routine, or are an experienced MASM user tired of waiting months to assemble large files, ***FAST ASSEM-86*** will bring the excitement back to assembly language.

***FAST ASSEM-86* IS MUCH FASTER:**

- How fast is ***FASM***™? The graph below shows relative assembly times for a 48K source file. For large files like this we blow MASM's doors off at 3 times their speed. For smaller 8K files we positively vaporize them at 6 times their speed.

FASM™ (110 sec.)
MASM (340 sec.)

- ***FAST ASSEM-86*** is faster for the following reasons: **(1)** Written entirely in assembly language (unlike MASM). **(2)** Editor, assembler and source file always in memory so you can go instantly from editing to assembling and back. **(3)** Eliminates the time needed to LINK programs. Executable .COM files can be created directly. (Also creates .OBJ files compatible with the IBM linker).

***FAST ASSEM-86* IS EASIER TO USE:**

FASM includes many other features to make your programming simpler.

- Listings are sent directly to screen or printer. Assemblies can be single stepped and examined without having to leave the editor.
- Access the built in cross reference utility from the editor.
- Full support of 186 and 286 (real mode) instructions.
- Both Microsoft and 8087 floating point formats are supported. 8087 and 287 instructions supported directly without macros for faster assembly.
- Calculator mode: Do math in any radix even using symbols from the symbol table.
- Direct to memory assembly feature lets you test execute your code from editor.
- Coming soon: A coordinated symbolic debugger.

COMPATIBILITY: ***FASM*** is source code compatibile with MASM and supports macros, records and structures.

Introductory Price $49
With .OBJ Capability $99

Speedware™

Dealer inquires welcome
118 Buck Circle
Folsom, CA 95630
(916) 988-7426

IBM, Turbo Pascal, Microsoft trademarks of IBM Corp., Borland Intern., Microsoft Corp.

Circle **no. 59** on reader service card.

Epsilon

The Emacs-Like Text Editor For Programmers Who Don't Like to Wait!!

State of the Art Text Editor

Epsilon is an exciting new text editor designed to make programmers more productive. Epsilon is faster than Brief, faster than Mince, faster than Gosling Emacs, *and faster than the editor you're using now!*

Concurrent Processes!

Epsilon lets you compile while you edit! You can run compilers, assemblers, linkers, and almost any other program that isn't screen oriented, all under Epsilon's control, while you edit your files!

With Epsilon you don't wait for programs like compilers to finish. Use Epsilon's concurrent process command, and while the compiler runs, you can continue to examine and edit files. Any errors in the compilation are displayed immediately, and Epsilon gives you the opportunity to correct them **while the compiler continues to run.** *With Epsilon, you're finished correcting errors when other editors first let you start.*

Powerful Commands

Epsilon has over 125 commands instantly available. Epsilon can manipulate words, sentences, and paragraphs easily. Epsilon will automatically save text you have deleted in a "ring" of kill-buffers, so that you can retrieve it later. It will help you avoid syntax errors by displaying matching parentheses. *And best of all, Epsilon's macros let you define your own commands, which can be loaded automatically each time you start Epsilon.*

Speed with No Limits.

Epsilon reads and writes files 25% to 600% faster than competing editors. From its convenient keyboard macros to its facility that completes the names of commands, files and buffers, to its optimized incremental search, Epsilon has been designed for programming ease and speed.

There's no limit to the number or the size of buffers you can have. Each buffer can hold a different file, or different versions of the same file. You can create as many windows as will fit on the screen, and display different buffers in each. And should you run out of memory, Epsilon will create and automatically utilize a swap file.

Speed Comparison (In Seconds) with Other Editors

	Epsilon	Brief	Mince	Emacs
Start-up	2.60	4.11	1.43	24.93
Read 21K file	1.06	1.33	8.95	7.52
Write 21K file	2.11	14.30	6.05	7.95
Next Screen	.19	.24	1.33	1.80
String Search	3.85	7.04	4.49	8.41
I-Search	3.85	--	--	8.73
First Help	8.30	12.33	--	--
Other Helps	.20	11.64	--	--

Epsilon runs on IBM PC's, XT's, AT's and compatibles with PC-DOS 2.0 or above and requires 192K of memory.

Epsilon's price is only $195.00.
ALL MAJOR CREDIT CARDS ACCEPTED.

Lugaru Software, Ltd.
5227 Fifth Avenue, Suite 12 / P. O. Box 110037
Pittsburgh, Pa. 15232

(412) 621-5911

Circle **no. 24** on reader service card.

Listing Three

```
0234 CDC902             CALL    PRCRLF          ; If equal, do CR/LF, reset
                                                ;  the col counter to zero,
                                                ;  increment line counter.
0237 3A7B04             LDA     INBUFPT         ; Set buffer pointer back
023A 3D                 DCR     A               ;  to ignore this char.
023B 327B04             STA     INBUFPT
023E F1                 POP     PSW
023F C3D301             JMP     PRINTLOOP       ; Reset stack and go back
                                                ;  thru read loop

0242 F1         NOTMAX: POP     PSW

0243 FE09       CKTAB:  CPI     TAB
0245 CC8602             CZ      TABTOSP         ; If TAB, expand it to spaces
0248 F5                 PUSH    PSW             ; Save data byte now in accum
0249 3A7D04             LDA     COLCNT
024C 3C                 INR     A               ; Increment column counter
024D 327D04             STA     COLCNT
0250 F1                 POP     PSW             ; Restore data for print

0251 CD2E03     GOPRNT: CALL    PCHAR
0254 C3D301             JMP     PRINTLOOP

0257 CDC902     DONE:   CALL    PRCRLF          ; Print CR/LF to wrapup
025A C9                 RET
                ;
                ;
                ; **********     SUBR GETNB     **********
                ;
                ;       Gets the next byte from memory or EOF flag if done
                ;
                ;       EXIT:   A = Next file byte fetched from memory
                ;               F = Carry set if End-of-File

025B 3A7B04     GETNB:  LDA     INBUFPT         ; Get current buff rel ptr
025E FE80               CPI     BUFF            ; Is it to end of rec yet?
0260 C27302             JNZ     GET             ; Get next byte if not.

                ;       Read a 128-byte record from disk

0263 115C00             LXI     D,FCB           ; DE gets FCB adr. Reads 128
0266 0E14               MVI     C,READF         ;  bytes into mem starting
0268 CD0500             CALL    BDOS            ;  at BUFF adr
026B B7                 ORA     A               ; A=0 if record read OK, so
026C CA7302             JZ      GET             ;  go get the info.
                                                ; Else if A#0, then is EOF.
026F 37         EOF:    STC                     ; Set carry to indicate EOF.
0270 C38502             JMP     GETEND

                GET:    ; Read the byte at BUFF + Reg A's relative offset

0273 5F                 MOV     E,A             ; DE contains rel position of
0274 1600               MVI     D,0             ;  pntr into disk buff space
0276 3C                 INR     A
0277 327B04             STA     INBUFPT         ; Incr and save new pointer.
027A 218000             LXI     H,BUFF          ; Start addr of buffer space
027D 19                 DAD     D               ;  plus rel offset ---> HL
027E 7E                 MOV     A,M             ; Get byte ptd to by HL
027F FE1A               CPI     CNTLZ           ; Check for EOF within record
0281 CA6F02             JZ      EOF             ;  & exit with carry bit set

0284 B7                 ORA     A               ; Reset carry bit
0285 C9         GETEND: RET
                ;
                ;
                ; **********     SUBR TABTOSP     **********
                ;
                TABTOSP:        ; Converts a tab to every NRSPCS columns

0286 3A7D04             LDA     COLCNT          ; Which column are we in?
0289 DE03               SBI     SKIP            ; Make relative offset
028B 5F                 MOV     E,A             ; Put it in E
028C 1608               MVI     D,NRSPCS
```

(Continued on page 86)

Listing Three

```
028E CDB002            CALL   MODULO          ; Do (COLCNT)mod NRSPCS
                                              ; Remainder is in Reg C
0291 3E08              MVI    A,NRSPCS
0293 91                SUB    C               ; A has total # spaces to do
0294 3D                DCR    A               ; Pickup space after RET
0295 4F                MOV    C,A             ; C has # spaces to do now
0296 FE01              CPI    1               ; Set sign if need less than
0298 FAAD02            JM     ENDTAB          ;  one space printed.

029B 3E20     SPACES:  MVI    A,SPC
029D C5                PUSH   B
029E CD2E03            CALL   PCHAR
02A1 C1                POP    B
02A2 3A7D04            LDA    COLCNT          ; Keep column count current
02A5 3C                INR    A               ;  while sending spaces
02A6 327D04            STA    COLCNT
02A9 0D                DCR    C
02AA C29B02            JNZ    SPACES          ; Loop until enough spaces

02AD 3E20     ENDTAB:  MVI    A,SPC           ; Print the last space when
                                              ;  get to GOPRNT
02AF C9                RET
              ;
              ;
              ; **********     SUBR MODULO     **********
              ;
              ;        Does division of 2 8-bit numbers: (Reg E)/(Reg D)
              ;        Result in Reg H
              ;        Remainer in Reg C = (Reg E) mod (Reg D)
              ;
02B0 210800   MODULO:  LXI    H,00001000B
02B3 0E00              MVI    C,0
02B5 7B       NEWBIT:  MOV    A,E
02B6 17                RAL
02B7 5F                MOV    E,A
02B8 79                MOV    A,C
02B9 17                RAL
02BA 92                SUB    D
02BB D2BF02            JNC    NOADD
02BE 82                ADD    D
02BF 4F       NOADD:   MOV    C,A
02C0 3F                CMC
02C1 7C                MOV    A,H
02C2 17                RAL
02C3 67                MOV    H,A
02C4 2D                DCR    L
02C5 C2B502            JNZ    NEWBIT
02C8 C9                RET
              ;
              ;
              ; **********     SUBR PRCRLF     **********
              ;
              PRCRLF:         ; Prints a CR/LF pair

02C9 F5                PUSH   PSW
02CA AF                XRA    A               ; Zero accum
02CB 327D04            STA    COLCNT          ; Set column counter to zero
02CE 3A7E04            LDA    LINECNT
02D1 3C                INR    A               ; Increment line counter
02D2 327E04            STA    LINECNT
02D5 3E0D              MVI    A,CR
02D7 CD2E03            CALL   PCHAR
02DA 3E0A              MVI    A,LF
02DC CD2E03            CALL   PCHAR
02DF F1                POP    PSW
02E0 C9                RET
              ;
              ;
              ; **********     SUBR BCDTOASC     **********
              ;
```

```
                ;       Converts binary value into two ASCII-hex characters
                ;
                ;       ENTRY:  Reg A   = 2 packed-BCD characters
                ;                   DE = Adr of where to store ASCII answers
                ;
                ;       EXIT:     (DE) = MS char
                ;               (DE+1) = LS char
                ;
                BCDTOASC:
02E1 F5                 PUSH    PSW
02E2 0F                 RRC                     ; Put 4 MSB's into 4 LSB's
02E3 0F                 RRC
02E4 0F                 RRC
02E5 0F                 RRC
02E6 E60F               ANI     0FH             ; Mask out top 4 bits
02E8 C630               ADI     '0'             ; Add offset to make  ASCII
02EA 12                 STAX    D               ; Save the result
02EB F1                 POP     PSW             ; Get the next character
02EC E60F               ANI     0FH
02EE C630               ADI     '0'             ; Make ASCII
02F0 13                 INX     D
02F1 12                 STAX    D               ; Save it in next memory loc
02F2 C9                 RET
                ;
                ;
                ; **********      SUBR NTOLIT     **********
                ;
                ;       Converts packed-BCD number (1 to 12) to its equiv
                ;       literal string and saves in specified memory.
                ;
                ;       ENTRY:  A  =  2 packed-BCD numbers from clock
                ;               DE =  Adr of where to store literal results
                ;               HL =  Pntr to 10-char literal strings
                ;
02F3 47         NTOLIT: MOV     B,A             ; Save data
02F4 E610               ANI     00010000B       ; See if have tens digit
02F6 CAFF02             JZ      SMALLNR         ; Jump if not
02F9 78                 MOV     A,B
02FA C60A               ADI     10              ; Add 10 to low nibble so
02FC C30003             JMP     BINARY          ;  it's binary now.
02FF 78         SMALLNR:MOV     A,B

0300 E60F       BINARY: ANI     0FH             ; Mask off top nibble
0302 D601               SUI     1               ; Make number 0 to 11 max
0304 CA0F03             JZ      MOVE            ; At first already

0307 010A00             LXI     B,10            ; 10 literals in word
030A 09         NEXT:   DAD     B               ; Add it to HL for next word
030B 3D                 DCR     A               ;  until get to proper one.
030C C20A03             JNZ     NEXT

030F 010A00     MOVE:   LXI     B,10            ; Cnt to move 10 letters only

    0312 7E         MOVDAT: MOV     A,M             ; HL pts to desired wor
0313 12                 STAX    D               ; DE pts to destination mem.
0314 13                 INX     D
0315 23                 INX     H
0316 0D                 DCR     C               ; Proper literal word now
0317 C21203             JNZ     MOVDAT          ;  moved to memory to print

031A C9                 RET
                ;
                ;
                ; **********      SUBR LISTIT     **********
                ;
                ;       Lists to printer until '$' encountered.
                ;
                ;       ENTRY:  HL  =  string address
                ;
031B 7E         LISTIT: MOV     A,M             ; Get the letter to be sent
031C FE24               CPI     '$'             ; End of text yet?
031E CA2D03             JZ      LISTEND
0321 E5                 PUSH    H               ; Must retain HL!
0322 5F                 MOV     E,A
0323 0E05               MVI     C,LISTOUT
0325 CD0500             CALL    BDOS            ; Print the char
```

(Continued on next page)

Listing Three

```
0328 E1                 POP     H
0329 23                 INX     H
032A C31B03             JMP     LISTIT          ; Keep going until '$'
032D C9         LISTEND:RET
                ;
                ;
                ; **********     SUBR PCHAR     **********
                ;
                PCHAR:  ; Prints Reg A to console or printer

                IF TOLPTR
032E 0E05               MVI     C,LISTOUT       ; Output to printer
                ENDIF
                ;
                IF NOT TOLPTR
                        MVI     C,CONSOUT       ; Output to console
                ENDIF

0330 5F                 MOV     E,A
0331 CD0500             CALL    BDOS
0334 C9                 RET
                ;
                ;
                ; **********     SUBR PRINTIT   **********
                ;
                ;       ENTRY:  DE = Start address of string
                ;
0335 F5         PRINTIT:PUSH    PSW

0336 0E09               MVI     C,PRINTST
0338 CD0500             CALL    BDOS            ; Print string to console
033B F1                 POP     PSW
033C C9                 RET
                ;
                ;
                ; **********   CONSOLE MESSAGES    **********
                ;

033D 0D0A496E70   OPENERR:  DB      CR,LF,'Input file not found or not '
035B 7370656369             DB      'specified!',CR,LF,'$'
                ;
                ;
                ; ********** TIME/DATE/PGM HEADING **********
                ;
0368              TOPLINE:  DS      SKIP

036B              HOURS:    DS      2
036D 3A                     DB      ':'
036E              MINUTES:  DS      2
0370 3A                     DB      ':'
0371              SECONDS:  DS      2
0373 2020202020             DB      '              '
0380              DAY:      DS      10
038A 20                     DB      ' '
038B              DATE:     DS      2
038D 20                     DB      ' '
038E              MONTH:    DS      10
0398 3139                   DB      '19'
039A              YEAR:     DS      2
039C 2020202020             DB      '                 '
03AA              HEADER:   DS      TSIZE
03B6 0D0A0A0A24             DB      CR,LF,LF,LF,'$'
                ;
                ;
                ; **********     LITERALS        **********
                ;
03BB 53756E6461 LITDAY:     DB      'Sunday,   '
03C5 4D6F6E6461             DB      'Monday,   '
03CF 5475657364             DB      'Tuesday,  '
03D9 5765646E65             DB      'Wednesday,'
03E3 5468757273             DB      'Thursday, '
03ED 4672696461             DB      'Friday,   '
```

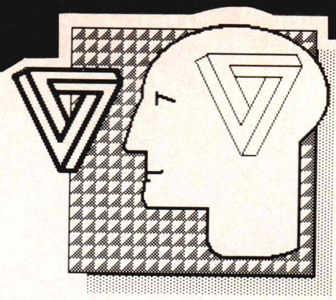
(Continued on page 90)

Listing Three

```
03F7 5361747572              DB       'Saturday, '

0401 4A616E7561 LITMON:      DB       'January   '
040B 4665627275              DB       'February  '
0415 4D61726368              DB       'March     '
041F 417072696C              DB       'April     '
0429 4D61792020              DB       'May       '
0433 4A756E6520              DB       'June      '
043D 4A756C7920              DB       'July      '
0447 4175677573              DB       'August    '

0451 5365707465              DB       'September '
045B 4F63746F62              DB       'October   '
0465 4E6F76656D              DB       'November  '
046F 446563656D              DB       'December  '

                ;
                ; **********      VARIABLES       **********
                ;

0479            OLDSTK:    DS     2              ; Stack pointer to get back
                                                 ;  to the CCP from this pgm.

047B            INBUFPT:   DS     2              ; Input buffer pointer

047D            COLCNT:    DS     1              ; Current length of line

047E            LINECNT:   DS     1              ; Keep count of lines printed
                                                 ;  on the page

                ;
                ; **********      STACK LOCATION      **********

047F                       DS     32             ; Reserve 16-level stack
                NEWSTK:

049F                       END
```

End Listings

REALIZABLE FANTASIES



by Michael Swaine and Bob Albrecht

We announced this column of imagined and realizable projects in February in a piece called "Tiny Hackers" and actually launched it in March with Richard Stallman's "GNU Manifesto." But the point of writing about realizable fantasies is to encourage their realization, so this month we present progress reports on the fantasies presented to date.

Liberating the Mac

In January we published, though not in this column, Tom Lafleur's illustrated guide to fattening your Macintosh. Tom received over a hundred calls and letters shortly after the issue came out. By the time of the Macworld show, several people had set up business dedicated to Mac-fattening, at least some of which seem to have been spin-offs from Tom's article. Despite our warnings that doing it yourself voided your warranty, risked frying your Mac and was a tedious process, a lot of you opened your Macs and went at it. We think that one reason for the article's popularity is that a lot of you believe that Apple's decision to build the Mac as a closed system was a mistake, and one you're willing to rectify. ("Apple just ain't the company it once Woz."—Laran Stardrake.)

In April we followed up on this belief by reporting in this column on various ways in which people could be said to be liberating the Mac, including Lee Felsenstein's Hacker's Mac (aka Hackintosh), a project for learning about the Mac by redesigning it; and Jack Tramiel's Atari ST (aka Jackintosh), which, running DRI's GEM environment, may bring Maclike capabilities to the rest of us. As this goes to press, Atari has just previewed the ST in Germany, DRI has announced the first applications for GEM and Lee thinks that Tramiel may be doing some of the Hackintosh team's work for it, though he hasn't abandoned the project. Contact Lee at Golemics, 2600 Tenth Street, Berkeley CA 94710, (415) 486-8344.

Also in April, we described Steve Jasik's MacNosy disassembler for the Mac, but failed to give a full address. It's Free the ROM 64, 343 Trenton Way, Menlo Park CA 94025, and his telephone number is (415) 322-1386. Steve has fixed some bugs in Nosy and is now supplying a fast sort routine with the program. MacNosy's opening screen echos the opening lines of the cult classic television program The Prisoner:

Who are you?	I am number two.
Who is number one?	You are number six.
What do you want?	Information.

GNU Manifestations

As last month's letters indicated, reaction to March's Realizable Fantasy, a proposal by Richard Stallman to develop a free operating system that provides the capabilities of Unix, has been emotional. Stallman's mail has been overwhelmingly supportive; he's received many offers of help, and he called to offer us quarterly updates on the project's progress. We'll probably print the first of these next month. Stallman is reachable at 166 Prospect St., Cambridge MA 02139.

Tiny Hackers

Dragonsmoke, Bob Albrecht's newsletter, is the place to follow The Dragon's master plan to turn innocent children into programmers, and it seems that it will soon be carrying excerpts from Ron Jeffries' newsletter, the *Jeffries Report*. Dragonsmoke costs $12/year and you can get a sample issue by sending $1.00 or $.39 and an SASE to Dragonsmoke, P.O.Box 7627, Menlo Park CA 94026.

Check the June issue of *Rainbow* for a review of the CoCoMac, a low-budget way to get Maclike features. The same issue also contains information on CoCoMac RAM disk construction and a plan to get the Fujitsu dual-6809 machine distributed in the U.S. It's supposedly very powerful, even without the 68K you can drop in.

Subversives

In a guest essay in this space in May, Resident Intern Dave Cortesi gave his vision of this magazine, a realizable fantasy for *DDJ*: that *DDJ* is and should always be subversive. He said, "As a good revolutionary, I must believe that computer use is for everybody, and I really do. But programming the computers so they can be used is almost certainly a job for specialists. Whose specialists?" Not the establishment's, Cortesi hopes, but those who want to promulgate ideas to all who can grasp them, those who do not see information exchange as a zero-sum game, those who see themselves involved in a revolution, those with a mischievous, subversive desire "to snatch the tools of the establishment and apply them in the public domain." We never ignore Cortesi's advice, and we hope you won't either. You have, through your contributions, the power to steer this magazine away from the safe shores and keep it pushing out recklessly on journeys of discovery.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 195.**

CP/M SYSTEM INVENTORY LIQUIDATION

Old operating systems never die, they get better with age! Multitech Electronics, a large manufacturer of complete computer systems has an inventory of CP/M computers that must go. These fully guaranteed high performance units are new factory packaged systems with the CP/M 2.2 operating system, 64K of RAM, dual floppy disk drives, and CBASIC programming language. You'll get more than what you paid for with complete systems at bargain prices.

FULL CP/M SYSTEM SPECS

	MIC-501	MIC-504
CPU	Z-80A	Z-80A
RAM MEMORY	64K	64K
STORAGE	500K BYTES	2M BYTES
SERIAL COMMUNICATIONS	DUAL RS—232C PORTS	DUAL RS-232C PORTS
PARALLEL COMMUNICATIONS	SINGLE PARALLEL PORT	SINGLE PARALLET PORT
PRICE	$500	$700

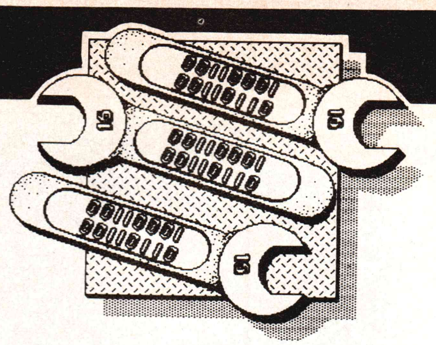
This low price also includes: full compatibility with the wide bank of CP/M software, CP/M 2.2 operating system with utilities and manuals, CBASIC 2 programming language, as well as a User's Manual and Service Manual. In addition you'll get a 30 day full warranty. And Multitech will also throw in a Liberty Freedom 100 terminal for $400.00 with the purchase of an MIC-500 system. Call Multitech today for more on your CP/M solutions. Cash or credit cards only. Limited quantities available.

Multitech ELECTRONICS INC.

195 WEST EL CAMINO REAL
SUNNYVALE, CALIFORNIA 94087
TELEX: 755042 MAC SUVL
IN CALIFORNIA, CALL: (408) 773-8400
OUTSIDE CALIFORNIA, CALL: (800) 538-1542

Circle **no. 42** on reader service card.

16-BIT SOFTWARE TOOLBOX



by Ray Duncan

MSDOS Installable Device Drivers

One of the more novel features added in Version 2 of MSDOS is the concept of "installable device drivers." This allows the user to attach new drivers for additional hardware devices or to supersede the system's existing built-in drivers by the simple expedient of putting the executable driver file on the boot disk and editing a line into the file CONFIG.SYS. This extremely powerful concept has made the lives of third-party mass storage device manufacturers much easier (previously they had to disassemble and patch the operating system to get their products to run).

But First . . . an Overview of Unix Device Drivers

Because the installable device drivers of MSDOS are patterned after Unix and much of the terminology used in the Microsoft driver documentation derives from Unix terminology, it is instructive to review the structure of the real McCoy. Much of the information presented here was gleaned from the article "Writing Device Drivers for Xenix Systems" by Jean McNamara, et al. (UniForum Conference Proceedings, January 1984).

Unix knows two types of devices: block and character. A block device is typically a mass storage medium such as a fixed or removable disk or magnetic tape drive. Such devices usually transfer data in chunks of fixed size, which are related (in the case of disks) to the characteristics and "format" of the physical media.

In contrast, a character device is a sequential device such as a CRT terminal or paper tape reader that supplies or accepts a stream of bytes. Although ideally the data stream of a character device has no inherent structure, in practice it usually is delimited by special control characters such as carriage returns, which the operating system also recognizes.

Under Unix, all drivers (whether controlling a block or character device) have the same general structure and contain two major parts: task time routines and an interrupt handler.

The task time routines are called at the time of the application program's (the task's) request for I/O. The Unix kernel transforms the request from its high-level, logical, device-independent character into addresses and parameters that are relevant to the physical device. When the driver's task time routines are executing, the application itself may be thought of as being in control of the system and active, although it is running in privileged or kernel mode.

The interrupt handler is entered asynchronously when the corresponding device generates a hardware interrupt. Interrupts are issued when an I/O operation has either finished or been terminated due to a hardware fault. If the system or the device does not support hardware interrupts, the interrupt handler is entered when the kernel, polling the device periodically, determines that it is no longer busy. Typically, the task requesting I/O from the device is inactive when the interrupt handler receives control; indeed, the task is usually in a suspended state pending I/O completion, and some other application is active and in control as far as the operating system is concerned.

A Unix block device driver comprises some tables and five major routines:

- *Init* is called once when the system is booted and the driver is first loaded into memory. It checks for the existence of the physical device and initializes it properly for future I/O, if present.
- *Open* is called by the Unix kernel when a user tries to access a file on the device. This routine should complete any initialization not performed by *Init* to prepare for I/O; it may be part of a mount sequence for removable media.
- *Close* is called by the Unix kernel to reset any flags or variables set by *Open* and to ensure that pending operations are completed. A typical action would be to flush buffered data to the physical device. This routine may be part of a dismount sequence for removable media.
- The *Strat* routine, an abbreviation for "strategy," is called by the Unix kernel when the application issues an I/O request for the block device. *Strat* is called with the address of a buffer header or parameter list, which defines the device unit number, request type (read, write, or format), where to find the data (logical block number), how much data it should transfer, and the memory address of its buffer. *Strat*'s responsibility is to validate the request then place it on the request queue for that device. Finally, it calls the internal routine *Start* (see below).
- *Intr* is called by the Unix kernel when the block device issues an interrupt. It is responsible for determining that the interrupt signal is valid (Was an operation on this device really in progress and are the appropriate completion flags set on the controller?). It determines success or failure status for the last operation by interrogating the controller and passes the appropriate flags back to the kernel. Finally, if additional requests are pending, it calls *Start* to try to initiate another I/O operation.

Start is an internal driver routine called by *Strat* or *Intr*; it removes the highest priority request from the request queue and initiates the I/O. If the device is busy or if no requests are waiting, *Start* simply exits.

The body of a Unix character device driver is similar to that of a block device driver. It is made up of seven major routines:

- *Init*, *Open*, and *Close* are similar in function and intent to those of the same name in a block device driver.
- *Read* transfers data from the device driver's input buffer to the application's buffer.
- *Write* transfers data from the application's buffers to the device driver's output buffers; it also starts output to the device if it is idle. If the output buffers are full, *Write* suspends the requesting process until they have emptied.
- *Ioctl* provides direct communication between the application and the driver. It allows the application to ask for device-specific information or to set the values of the driver's internal flags and variables (such as baud rate or number of stop bits for a serial I/O port).
- *Intr*, the asynchronous part of the driver, is called by the kernel when the device issues a hardware interrupt. This procedure performs the actual data transfer between the physical device's hardware controller and the driver's input/output buffers.

MSDOS Device Drivers

MSDOS installable device drivers bear a strong resemblance to Unix device drivers, both structurally and functionally. Like Unix, they fall into two major classes.

Character device drivers control devices that perform I/O one byte at a time, similar to a traditional TTY terminal. MSDOS has built-in drivers for the console device, serial port, and list device, named CON, AUX, and PRN, respectively. You can access these drivers using the traditional CP/M-like character I/O calls, or you can open them by name, like a file for input and output, using the new "handle" function calls of MSDOS 2.0 and above. A character device driver can support one hardware unit.

Block device drivers control random access storage devices such as flexible disk drives or fixed disks. A block device driver can support more than one hardware unit and/or may map a single physical unit onto two or more logical drives. Each device unit for a given block driver is assigned a drive designator (such as A:, B:, etc.); the first drive letter for a given block device driver is determined by that driver's position in the overall chain of drivers.

Once a driver is written and assembled, you may load and link it into the operating system simply by placing an entry of the form

```
device=filename.ext
```

in the CONFIG.SYS file on the system boot disk. You can override the default system driver for a character device with an installed driver simply by giving it the same logical device name in the device header. When processing an I/O request, DOS always scans the list of installed drivers before the default devices and takes the first match.

Structure of an MSDOS Device Driver

MSDOS device drivers always have an ORIGIN of zero but are otherwise assembled, linked, and converted into an executable module as though they were COM or EXE files. A device driver consists of three major parts.

A Device Header (Table 1, below) contains the linkage to the next driver in the chain, a set of attribute flags (Table 2, below) for the device, offsets to the executable strategy and interrupt routines for the device, and the logical device name (if it is a character device such as PRN or COM1). The linkage address of the last driver in a file is

Offset	Contents
0	Offset, pointer to next device header
2	Segment, pointer to next device header
4	Device attribute word (see Table 2)
6	Pointer to device strategy code (offset)
8	Pointer to device interrupt code (offset)
10	Logical name (8 bytes) if character device or number of units (1 byte) if block device followed by 7 bytes that may contain a name or nothing at all

Table 1
Device driver header.

Bit	Meaning
15	=1 if character device =0 if block device
14	=1 if IOCTL is supported
13	=1 if non-IBM format (block only)
12	#
11	=1 if open/close/RM is supported*
10	#
9	#
8	#
7	#
6	#
5	#
4	#
3	=1 if current clock device
2	=1 if current NUL device
1	=1 if current standard output device
0	=1 if current standard input device

Currently undefined and should be 0
* DOS 3.0 and above only; should be 0 for DOS 2.x

Table 2
Device attribute word, found in device driver header; only bits 11, 13, and 14 have significance on block devices.

always set to −1 when the driver is created; the drivers are chained together, and the linkage fields are updated by MSDOS at system initialization.

The Strategy Routine for the device is entered by DOS via a Far Call when the driver is first loaded and installed and whenever an application program issues an I/O request for the device. DOS uses ES:BX to pass the Strategy routine a double-word pointer to a data structure, which is called a Request Header; this structure contains information about the type of operation to be performed. According to MSDOS conventions, the Strategy code never actually performs an I/O operation but simply saves the pointer to the Request Header.

The last and most complex part of a device driver is the misnamed Interrupt Routine. This code implements the device driver proper; it performs the actual operation based on the function code and other information passed in the Request Header. Status and completion information may be passed back to DOS in the same Header.

When an I/O request is issued, the Interrupt Routine entry point is called by DOS immediately after the call to the Strategy Routine. Unlike in Unix, the Interrupt Routine is never entered asynchronously (as is the case on an I/O completion interrupt). The division of function between the Strategy and Interrupt Routines is completely artificial, at least under the currently available nonmultitasking versions of MSDOS.

```
;
; MS-DOS Request Header structure definition
;
Request   struc            ; request header template
                              structure
Rlength   db ?             ; 0  length of request header
Unit      db ?             ; 1  unit number for this request
Command   db ?             ; 2  request header's command
                                code
Status    dw ?             ; 3  driver's return status word
Reserve   db 8 dup (?)     ; 5  reserved area
Media     db ?             ; 13 media descriptor byte
Address   dd ?             ; 14 memory address for transfer
Count     dw ?             ; 18 byte/sector count value
Sector    dw ?             ; 20 starting sector value
Request   ends             ; end of request header template
```

Table 3
Format of request header. Only the first 13 bytes are common to all driver functions; the number and definition of the following bytes vary depending on the function type. The structure shown here is the one used by the read and write subfunctions of the driver.

Bit(s)	Significance
15	Error
10–14	Reserved
9	Busy
8	Done
0–7	Error code if bit 15 = 1

Table 4
Return status word of request header.

The Request Header

The MSDOS BDOS uses a data structure called the Request Header to give the driver the necessary information to perform an I/O operation. The address of the Request Header is passed to the Strategy Routine and saved in a local variable; the Interrupt Routine, which is called immediately afterward, uses this address to access information from the Header.

The first 13 bytes of the Request Header are the same for all device driver functions and therefore are referred to as the "static" portion of the Header. The number and contents of the following bytes vary according to the type of function requested (Table 3, at left). The Request Header's primary components are a Command Code that selects a driver subfunction (such as Read, Write, or Status) and a Return Status word that informs the BDOS about the driver's success with the request I/O operation (Table 4, below left, and Table 5, page 98). Other information passed in the Header to the driver includes minor unit numbers, transfer addresses, sector or byte counts, and so on.

When an I/O function is completed, the device driver uses fields in the Request Header to pass status, sector or byte counts, and other information back to the operating system.

The Driver Command Code Routines

The Command Code for the requested driver subfunction is passed in the third byte of the Request Header. The Interrupt Routine extracts it from the Request Header using the double-word pointer saved during the call to the Strategy Routine. Typically, the Command Code is used as an index into a jump table that points to the proper service code.

In the descriptions below, RH refers to the Request Header whose address was passed to the Strategy Routine in ES:BX. DWORD refers to a long address, of which the first two bytes contain the offset and the last two bytes contain the segment.

Function 0—Driver Initialization

This initialization code for the driver is called only once, when the driver is loaded. It is responsible for performing any necessary hardware initialization of the device, setup of interrupt vectors, and so on. It returns the address of the start of free memory after the driver, so that DOS knows where it can load the next installable driver. If it is initializing a block device driver, it must also return the number of units and the address of the BIOS parameter block (BPB) pointer array; if all units are the same, all pointers in the array can point to the same BPB.

Technical Product Information

FREE
For The Asking

See something you'd like to learn more about? Need more details?

DR. DOBB'S JOURNAL **Reader Service Card**

Name ______________________ Phone ____________

Address ______________________________

City/State/Zip ______________________________

Expiration Date: Oct. 31, 1985 July 1985 #106

Please circle one for each category:

I. My firm or department is a:
- A. Systems Integrator/House
- B. Software Dev. Firm
- C. Hardware OEM or Manuf.
- D. DP, MIS or Data Service
- E. Consulting Firm
- F. Eng. or Science Lab.
- G. Other

II. My job function is:
- H. Company Mgmt./Admin.
- J. Computer Systems Mgt.
- K. Programmer/Technical Staff
- L. Consultant
- M. Engineering Mgmt. or Staff
- N. Scientific Mgmt. or Staff
- O. Other

III. Number of employees in my firm:
- 1. Less than 10
- 2. 10—99
- 3. 100—499
- 4. 500—9,999
- 5. 10,000—or More

IV. This inquiry is for:
- P. Immediate Purchase
- Q. Future Project

V. Purchasing Authority (check all that apply)
- R. Recommend or Specify
- S. Final Decision-Maker
- T. No Influence

VI. I advise others about computers, on the average:
- 6. More Than Once-A-Day
- 7. Once-A-Day
- 8. Once-A-Week
- 9. Not At All

VII. I design/write software professionally
- U. Yes
- V. No

VIII. I buy computer products through: (check all that apply)
- W. Retail Stores
- X. Mail Order Houses
- Y. On-site Direct Salespeople
- Z. All of the Above

IX. I am currently a subscriber:
- A. Yes
- B. No

Check each advertisement for corresponding number and circle below:

001	011	021	031	041	051	061
002	012	022	032	042	052	062
003	013	023	033	043	053	063
004	014	024	034	044	054	064
005	015	025	035	045	055	065
006	016	026	036	046	056	066
007	017	027	037	047	057	067
008	018	028	038	048	058	068
009	019	029	039	049	059	069
010	020	030	040	050	060	070
071	081	091	101	111	121	131
072	082	092	102	112	122	132
073	083	093	103	113	123	133
074	084	094	104	114	124	134
075	085	095	105	115	125	135
076	086	096	106	116	126	136
077	087	097	107	117	127	137
078	088	098	108	118	128	138
079	089	099	109	119	129	139
080	090	100	110	120	130	140
141	151	161	171		**Articles**	
142	152	162	172	181	191	201
143	153	163	173	182	192	202
144	154	164	174	183	193	203
145	155	165	175	184	194	204
146	156	166	176	185	195	205
147	157	167	177	186	196	206
148	158	168	178	187	197	207
149	159	169	179	188	198	208
150	160	170	180	189	199	209
				190	200	210

☐ **Please send me a one year subscription to Dr. Dobb's Journal at $25.00 and bill me later.**

Note:
For quicker, more effective processing of your inquiry, please provide responses to ALL requested information.

Information that's
- *Current*
- *In-depth*
- *Directed to you on specific products & services*

FREE

Send us your POSTAGE-PAID card Today!

DR. DOBB'S JOURNAL **Reader Service Card**

Name ______________________ Phone ____________

Address ______________________________

City/State/Zip ______________________________

Expiration Date: Oct. 31, 1985 July 1985 #106

Please circle one for each category:

I. My firm or department is a:
- A. Systems Integrator/House
- B. Software Dev. Firm
- C. Hardware OEM or Manuf.
- D. DP, MIS or Data Service
- E. Consulting Firm
- F. Eng. or Science Lab.
- G. Other

II. My job function is:
- H. Company Mgmt./Admin.
- J. Computer Systems Mgt.
- K. Programmer/Technical Staff
- L. Consultant
- M. Engineering Mgmt. or Staff
- N. Scientific Mgmt. or Staff
- O. Other

III. Number of employees in my firm:
- 1. Less than 10
- 2. 10—99
- 3. 100—499
- 4. 500—9,999
- 5. 10,000—or More

IV. This inquiry is for:
- P. Immediate Purchase
- Q. Future Project

V. Purchasing Authority (check all that apply)
- R. Recommend or Specify
- S. Final Decision-Maker
- T. No Influence

VI. I advise others about computers, on the average:
- 6. More Than Once-A-Day
- 7. Once-A-Day
- 8. Once-A-Week
- 9. Not At All

VII. I design/write software professionally
- U. Yes
- V. No

VIII. I buy computer products through: (check all that apply)
- W. Retail Stores
- X. Mail Order Houses
- Y. On-site Direct Salespeople
- Z. All of the Above

IX. I am currently a subscriber:
- A. Yes
- B. No

Check each advertisement for corresponding number and circle below:

001	011	021	031	041	051	061
002	012	022	032	042	052	062
003	013	023	033	043	053	063
004	014	024	034	044	054	064
005	015	025	035	045	055	065
006	016	026	036	046	056	066
007	017	027	037	047	057	067
008	018	028	038	048	058	068
009	019	029	039	049	059	069
010	020	030	040	050	060	070
071	081	091	101	111	121	131
072	082	092	102	112	122	132
073	083	093	103	113	123	133
074	084	094	104	114	124	134
075	085	095	105	115	125	135
076	086	096	106	116	126	136
077	087	097	107	117	127	137
078	088	098	108	118	128	138
079	089	099	109	119	129	139
080	090	100	110	120	130	140
141	151	161	171		**Articles**	
142	152	162	172	181	191	201
143	153	163	173	182	192	202
144	154	164	174	183	193	203
145	155	165	175	184	194	204
146	156	166	176	185	195	205
147	157	167	177	186	196	206
148	158	168	178	187	197	207
149	159	169	179	188	198	208
150	160	170	180	189	199	209
				190	200	210

☐ **Please send me a one year subscription to Dr. Dobb's Journal at $25.00 and bill me later.**

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT #27346, PHILADELPHIA, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS
Dr. Dobb's Journal
P.O. BOX 13851
PHILADELPHIA, PA 19101

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT #27346, PHILADELPHIA, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS
Dr. Dobb's Journal
P.O. BOX 13851
PHILADELPHIA, PA 19101

Dr. Dobb's Stands Apart

Take a good, hard look at the crowded computer magazine market. If you're a serious microcomputerist, one magazine stands apart. ***Dr. Dobb's Journal.***

Dr. Dobb's is not for everyone. It is written by and for expert programmers, and it's the oldest and most technically sophisticated microcomputer publication available. Since 1976, ***Dr. Dobb's Journal*** has been the unchallenged leading source of software tools for advanced programmers.

With the industry moving forward faster than ever, you need to stay a step ahead. ***Dr. Dobb's*** sets the pace with:

- regular columns on C, Unix, CP/M and 16-bit software;
- algorithms and problem solving;
- lively discussions of fundamental software issues;
- inside information on commercial languages and operating systems;
- tips on advanced programming topics.

The information and valuable code contained in ***Dr. Dobb's*** makes each issue indispensable for serious computing professionals and enthusiasts. Don't miss a single issue of this valuable resource. Subscribe today. If you aren't fully satisfied, cancel your subscription and keep the first issue as a free sample.

To start your subscription,
fill out this coupon and return it to:

Dr. Dobb's Journal
P.O. Box 27809, San Diego, CA 92128

Yes! Please enter my subscription for 12 issues of:
Dr. Dobb's Journal for $25.

If I am not fully satisfied I will write "cancel" on my subscription invoice and keep the first issue as a free sample.

*Name*____________________

*Address*____________________

*City*____________________

*State*__________ *Zip*__________

☐ *Bill me later* ☐ *I've enclosed $25* *Charge my* ☐ *VISA* ☐ *M/C*

*Card No.*____________________

*Expiration*____________________

*Signature*____________________

Offer good in U.S. only 4005

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS
Dr. Dobb's Journal

If the initialization routine finds that the device is missing or defective and wants to abort without using memory, it should set the number of units to zero and set the ending address to CS:0000.

The operating system services that the initialization code can invoke at load time are limited; the code can call only MSDOS services 01–0CH and 30H. This is adequate to check the DOS version number and display a driver identification message, but not much else.

Many programmers position the initialization code at the end of the driver and return its address as the location of the first free memory; this allows the memory occupied by the code to be reclaimed after it is finished with its work.

This routine is called with:

RH+18	DWORD	Pointer to the character after the = on the CONFIG.SYS line that loaded driver; this information is read only
RH+22	BYTE	Drive letter for first unit of a block driver: 0=A, 1=B, etc. (MSDOS 3.x only)

This routine returns:

RH+3	WORD	Return status
RH+13	BYTE	Number of units (block devices only)
RH+14	DWORD	Address of first free memory above driver
RH+18	DWORD	BPB pointer array (block devices only)

Code	Error Definition
0	Write protect violation
1	Unknown unit
2	Drive not ready
3	Unknown command
4	CRC error
5	Bad drive request structure length
6	Seek error
7	Unknown media
8	Sector not found
9	Printer out of paper
10	Write fault
11	Read fault
12	General failure
13–14	Reserved
15	Invalid disk change (MSDOS 3.x)

Table 5
Driver error codes returned in bits 0–7 of return status word in request header.

Byte(s)	Contents
0–1	Bytes per sector
2	Sectors per allocation unit (must be power of 2)
3–4	Number of reserved sectors (starting at sector 0)
5	Number of file allocation tables (FATs)
6–7	Maximum number of root directory entries
8–9	Total number of sectors in media
10	Media descriptor byte
11–12	Number of sectors occupied by a single FAT

Table 6
Format of BIOS parameter block; a copy of this block is always found in the boot sector of an initialized disk.

Function 1—Media Check

The media check function is used on block devices only and should be a NOP in character device drivers. This routine is called first by BDOS for a block device transfer, passing the current media descriptor byte (Table 7, page 99). If feasible, the routine returns a code indicating whether the media has changed since the last transfer. This feature requires a machine that provides a door interlock hardware status flag or something similar. Determining the media change status improves performance because MSDOS does not need to reread the file allocation table (FAT) for each directory access.

This routine is called with:

RH+1	BYTE	Unit code
RH+13	BYTE	Media descriptor byte

This routine returns:

RH+3	WORD	Return status
RH+14	BYTE	Media change code: –1 Media has changed 0 Don't know if media changed 1 Media has not changed
RH+15	DWORD	Pointer to previous volume ID, if device attribute bit 11 = 1 and media has changed (MSDOS 3.x only)

Function 2—Build BIOS Parameter Block

The build BPB function is supported on block devices only and should be a NOP for character devices. The BDOS uses it to get a pointer to the valid BPB (Table 6, at left) for the current media. This routine is called when the media check routine returns a "Media has changed" code or when it returns a "Don't know if media changed" code and there are no dirty buffers (buffers with changed data that have not yet been written to disk). Thus this call indicates whether the media has legally changed. Under MSDOS 3.x, this function should also read the volume ID off the disk and save it.

The build BPB call also receives a pointer to a one-sector buffer in the address field of the request header. If the "non-IBM-format" bit in the device attribute word is

zero, the buffer contains the first sector of the FAT, including the media descriptor byte, and should not be altered by the driver. If the "non-IBM-format" bit is set, the buffer may be used as scratch space.

This routine is called with:

RH+1	BYTE	Unit code
RH+13	BYTE	Media descriptor byte
RH+14	DWORD	Buffer address

This routine returns:

RH+3	WORD	Return status
RH+18	DWORD	Pointer to new BPB

Function 3—I/O Control Read

This function allows control information to be passed directly from the application program to the device driver. It is called only if the IOCTL bit is set in the device header attribute word. No error check is performed on IOCTL I/O calls.

This routine is called with:

RH+1	BYTE	Unit code (block devices only)
RH+13	BYTE	Media descriptor byte
RH+14	DWORD	Transfer address
RH+18	WORD	Byte/Sector count
RH+20	WORD	Starting sector number (block devices only)

This routine returns:

RH+3	WORD	Return status
RH+18	WORD	Actual bytes or sectors transferred

Function 4—Read

This function transfers data from the device into the specified memory buffer. Under MSDOS 3.x, the routine can use the reference count of open files maintained by the open and close routines (functions 13 and 14) and the media descriptor byte to determine whether the media has changed illegally.

This routine is called with:

RH+1	BYTE	Unit code (block devices only)
RH+13	BYTE	Media descriptor byte
RH+14	DWORD	Transfer address
RH+18	WORD	Byte/Sector count
RH+20	WORD	Starting sector number (block devices only)

This routine returns:

RH+3	WORD	Return status
RH+18	WORD	Actual bytes or sectors transferred
RH+22	DWORD	Pointer to volume ID if error 0FH is returned (MSDOS 3.x only)

Function 5—Nondestructive Read

This function is available on character devices only. If an input status request returns a busy bit = 0 (characters waiting), the next character that would be read is returned to DOS but stays in the input buffer. This basically provides DOS with the capability to look ahead by one character.

This routine returns:

RH+3	WORD	Return status
RH+13	BYTE	Character

Function 6—Input Status

This function is available on character devices only and returns the current input status for the device. MSDOS

Media Descriptor Byte

Bit(s)	Significance
3–7	Always set (=1)
2	1=removable
	0=not removable
1	1=8 sector
	0=not 8 sector
0	1=2 sided
	0=not 2 sided

Current Valid MSDOS Descriptor Bytes (5¼-inch Disks)

0F9H	2 sided, 15 sector
0FCH	1 sided, 9 sector
0FDH	2 sided, 9 sector
0FEH	1 sided, 8 sector
0FFH	2 sided, 8 sector
0F8H	(fixed disk)

Table 7
Media descriptor byte of request header, assuming "non-IBM-format" bit in attribute word of device header is zero.

Addr	Attr	Str	Int	Type	Units	Name
00E3:0111	8004	0FD5	0FE0	C		NUL
0070:0148	8013	008E	0099	C		CON
0070:01DD	8000	008E	009F	C		AUX
0070:028E	8000	008E	00AE	C		PRN
0070:0300	8008	008E	00C3	C		CLOCK
0070:03CC	0000	008E	00C9	B	02	
0070:01EF	8000	008E	009F	C		COM1
0070:02A0	8000	008E	00AE	C		LPT1
0070:06F0	8000	008E	00B4	C		LPT2
0070:0702	8000	008E	00BA	C		LPT3
0070:0714	8000	008E	00A5	C		COM2

End of device chain.

Table 8
Example listing of device chain under MSDOS 2.1: "plain vanilla" PC with no hard disks or user device drivers.

assumes all character devices have a type-ahead buffer. If the device does not have a type-ahead buffer, it should always return a busy bit = 0 so MSDOS will not hang.

This routine returns:

RH+3	WORD	Return status Busy bit = 1 Read request goes to physical device Busy bit = 0 Characters already in device buffer; read request returns quickly

Function 7—Flush Input Buffers

This function is available on character devices only. It terminates all pending requests; i.e., the input buffer is emptied.

This routine returns:

RH+3	WORD	Return status

Function 8—Write

This routine transfers data from the specified memory buffer to the device. Under MSDOS 3.x, the routine can use the reference count of open files maintained by the open and close routines (functions 13 and 14) and the media descriptor byte to determine whether the media has changed illegally.

This routine is called with:

RH+1	BYTE	Unit code (block devices only)
RH+13	BYTE	Media descriptor byte
RH+14	DWORD	Transfer address
RH+18	WORD	Byte/Sector count
RH+20	WORD	Starting sector number (block devices only)

This routine returns:

RH+3	WORD	Return status
RH+18	WORD	Actual bytes or sectors transferred
RH+22	DWORD	Pointer to volume ID if error 0FH is returned (MSDOS 3.x only)

Function 9—Write with Verify

This routine transfers data from the specified memory buffer to the device. If feasible, a read-after-write verification of the data is performed to confirm that the data was written correctly. Otherwise, it is exactly like function 8.

Function 10—Output Status

This routine returns the current output status for the device. This function is available on character devices only.

This routine returns:

RH+3	WORD	Return status Busy bit = 1 Write request waits

for completion of current request
Busy bit = 0 Device idle; write request starts immediately

Function 11—Flush Output Buffers

This function is available on character devices only and terminates all pending output requests. The output buffer, if any, is emptied.

This routine returns:

RH+3	WORD	Return status

Function 12—I/O Control Write

This function allows control information to be passed directly from the driver to the application program. It is called only if the IOCTL bit is set in the device header attribute word. No error check is performed on IOCTL I/O calls.

This routine is called with:

RH+1	BYTE	Unit code (block devices only)
RH+13	BYTE	Media descriptor byte
RH+14	DWORD	Transfer address
RH+18	WORD	Byte/Sector count
RH+20	WORD	Starting sector number (block devices only)

This routine returns:

RH+3	WORD	Return status
RH+18	WORD	Actual bytes or sectors transferred

Function 13—Open

This function is available on MSDOS version 3.0 and above only and is called only if the open/close/RM bit is set in the device attribute word.

Block devices may use the open function to manage local buffering and to increment a reference count of the number of open files on a device.

Character devices can use this function to send a device initialization string, which in turn can be set by IOCTL Write. Note that the predefined CON, AUX, and PRN devices are always open.

This routine is called with:

RH+1	BYTE	Unit code (block devices only)

This routine returns:

RH+3	WORD	Return status

Function 14—Device Close

This function is available on MSDOS version 3.0 and above only and is called only if the open/close/RM bit is set in the device attribute word.

On block devices, this function can manage local buffering and decrement a reference count keeping track of the number of open files on the device; when the count reaches zero, all files have been closed, and the driver should flush buffers because the user may change disks.

On character devices, this function can send a device-dependent post-I/O string such as a form feed, which in turn can be set by an IOCTL Write. Note that the CON, AUX, and PRN devices are never closed.

This routine is called with:

RH+1	BYTE	Unit code (block devices only)

This routine returns:

RH+3	WORD	Return status

Function 15—Removable Media

This function is available on MSDOS version 3.0 and above only and is called only if the open/close/RM bit is set in the device attribute word and the device is a block type.

This routine is called with:

RH+1	BYTE	Unit code (block devices only)

This routine returns:

RH+3	WORD	Return status Busy bit = 1 Media is non-removable Busy bit = 0 Media is removable

DDJ

16-Bit Toolbox Listing (Text begins on page 94)

```
                name    driver
                page    55,132
                title   'DRIVER --- installable driver template'

;
; This is a "template" for a MS-DOS installable device driver.
; The actual driver subroutines are stubs only and have
; no effect but to return a non-error "done" status.
;
; Ray Duncan, April 1985
```

(Continued on next page)

```
                                     ; Laboratory Microsystems Inc.

      0000                           code    segment public 'CODE'

      0000                           driver  proc    far

                                             assume  cs:code,ds:code,es:code

      0000                                   org     0

      = 000F                         Max_Cmd equ     15                ; driver command code maximum
                                                                       ; 12 for MS-DOS 2.x,
                                                                       ; 15 for MS-DOS 3.x

      = 000D                         cr      equ     0dh               ; ASCII carriage return
      = 000A                         lf      equ     0ah               ; ASCII line feed
      = 0024                         eom     equ     '$'               ; end of message signal

                                             page
                                     ;
                                     ; Device Driver Header
                                     ;
      0000  FF FF FF FF              Header  dd      -1                ;link to next device,-1= end of list

      0004  8000                             dw      8000h             ;attribute word
                                                                       ;bit 15=1 for character devices

      0006  0056 R                           dw      Strat             ;device "Strategy" entry point

      0008  0061 R                           dw      Intr              ;device "Intrrupt" entry point

      000A  44 52 49 56 45 52                db      'DRIVER  '        ;char device name, 8 char, or
            20 20
                                                                       ;if block device, no. of units
                                                                       ;in first byte followed by
                                                                       ;7 don't care bytes

                                     ;
                                     ; local variables for use by driver
                                     ;
      0012  ????????                 RH_Ptr  dd      ?                 ; pointer to request header
                                                                       ; passed to Strat by BDOS

      0016  0D 0A 0A                 Ident   db      cr,lf,lf
      0019  45 78 61 6D 70 6C                db      'Example Device Driver 1.0'
            65 20 44 65 76 69
            63 65 20 44 72 69
            76 65 72 20 31 2E
            30
      0032  0D 0A 0A 24                      db      cr,lf,lf,eom

                                             page

                                     ;
                                     ; Driver Command Codes dispatch table
                                     ;
                                     ; The "Intr" routine uses this table and the Command Code
                                     ; supplied in the Request Header to transfer to the
                                     ; appropriate driver subroutine.

      0036                           Dispatch:

      0036  00BA R                           dw      Init              ;  0 = init driver into system
      0038  009C R                           dw      Media_Chk         ;  1 = media check on blk dev
      003A  009E R                           dw      Build_Bpb         ;  2 = build BIOS param block
      003C  00A0 R                           dw      Ioctl_Inp         ;  3 = I/O ctrl read from dev
      003E  00A2 R                           dw      Input             ;  4 = normal destructive read
      0040  00A4 R                           dw      Nd_Input          ;  5 = non-destructive read,no wait
      0042  00A6 R                           dw      Inp_Stat          ;  6 = return current input status
      0044  00A8 R                           dw      Inp_Flush         ;  7 = flush device input buffers
      0046  00AA R                           dw      Output            ;  8 = normal output to device
      0048  00AC R                           dw      Outp_Vfy          ;  9 = output with verify
      004A  00AE R                           dw      Outp_Stat         ; 10 = return current output status
```

(Continued on page 104)

 (Listing Continued, text begins on page 94)

```
004C  00B0 R                        dw      Outp_Flush      ; 11 = flush output buffers
004E  00B2 R                        dw      Ioctl_Outp      ; 12 = I/O control output
0050  00B4 R                        dw      Dev_Open        ; 13 = device open      (MS-DOS 3.x)
0052  00B6 R                        dw      Dev_Close       ; 14 = device close     (MS-DOS 3.x)
0054  00B8 R                        dw      Rem_Media       ; 15 = removeable media (MS-DOS 3.x)

                                    page
                            ;
                            ; MS-DOS Request Header structure definition
                            ;
                            ; The first 13 bytes of the Request Header are the same
                            ; for all Command codes and are termed the "Static" part of
                            ; the Header.  The number and meaning of the following bytes
                            ; vary depending on the Command code.
                            ;
                            ; The Request Header shown here applies to Read & Write functions.
                            ;

                            Request struc                   ; request header template structure

0000  ??                    Rlength db      ?               ; length of request header
0001  ??                    Unit    db      ?               ; unit number for this request
0002  ??                    Command db      ?               ; request header's command code
0003  ????                  Status  dw      ?               ; driver's return status word
0005     08 [               Reserve db      8 dup (?)       ; reserved area
               ??
                    ]

000D  ??                    Media   db      ?               ; media descriptor byte
000E  ????????              Address dd      ?               ; memory address for transfer
0012  ????                  Count   dw      ?               ; byte/sector count value
0014  ????                  Sector  dw      ?               ; starting sector value

0016                        Request ends                    ; end of request header template

                                    page

                            ; Device Driver "Strategy Routine"

                            ; Each time a request is made for this device, the BDOS
                            ; first calls "Strategy routine",  then immediately calls
                            ; the "Interrupt routine".

                            ; The Strategy routine is passed the address of the
                            ; Request Header in ES:BX, which it saves in a local
                            ; variable and then returns to BDOS.

0056                        Strat   proc    far
                                                            ; save address of Request Header
0056  2E: 89 1E 0012 R              mov     word ptr cs:[RH_Ptr],bx
005B  2E: 8C 06 0014 R              mov     word ptr cs:[RH_Ptr+2],es

0060  CB                            ret                     ; back to BDOS

0061                        Strat   endp

                                    page


                            ; Device Driver "Interrupt Routine"

                            ; This entry point is called by the BDOS immediately after
                            ; the call to the "Strategy Routine", which saved the long
                            ; address of the Request Header in the local variable "RH_Ptr".

                            ; The "Interrupt Routine" uses the Command Code passed in
                            ; the Request Header to transfer to the appropriate device
                            ; handling routine.  Each command code routine is responsible
                            ; for any necessary return information into the Request Header,
                            ; then transfers to Error or Exit to set the Return Status code.

0061                        Intr    proc  far
```

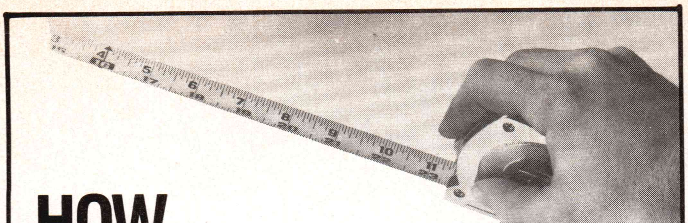
```
0061  50                 push    ax              ; save general registers
0062  53                 push    bx
0063  51                 push    cx
0064  52                 push    dx
0065  1E                 push    ds
0066  06                 push    es
0067  57                 push    di
0068  56                 push    si
0069  55                 push    bp

006A  0E                 push    cs              ; make local data addressable
006B  1F                 pop     ds

006C  C4 3E 0012 R       les     di,[RH_Ptr]     ; ES:DI = Request Header

                                                 ; get BX = Command Code
0070  26: 8A 5D 02       mov     bl,es:[di.Command]
0074  32 FF              xor     bh,bh
0076  83 FB 0F           cmp     bx,Max_Cmd      ; make sure its legal
0079  7F 06              jg      Unk_Command     ; too big, exit with error code
007B  D1 E3              shl     bx,1            ; form index to Dispatch table
                                                 ; and branch to driver routine
007D  FF A7 0036 R       jmp     word ptr [bx+Dispatch]

                         page


                 ; General collection of exit points for the driver routines.


0081             Unk_Command:                    ; Come here if Command Code too big.
0081  B0 03              mov     al,3            ; Sets "Unknown Command" error
                                                 ; code and "Done" bit.

0083             Error:                          ; Transfer here with AL = error code.
0083  B4 81              mov     ah,81h          ; Sets "Error" and "Done" bits.
0085  EB 03 90           jmp     Exit

0088  B4 01      Done:   mov     ah,1            ; Come here if I/O complete and
                                                 ; no error, sets "Done" bit only.



008A             Exit:                           ; General purpose exit point.
                                                 ; Transfer here with AX =
                                                 ; Return Status word to be
                                                 ; placed into Request Header.

008A  2E: C5 1E 0012 R   lds     bx,cs:[RH_Ptr]          ; set status
008F  89 47 03           mov     ds:[bx.Status],ax

0092  5D                 pop     bp              ;restore general registers
0093  5E                 pop     si
0094  5F                 pop     di
0095  07                 pop     es
0096  1F                 pop     ds
0097  5A                 pop     dx
0098  59                 pop     cx
0099  5B                 pop     bx
009A  58                 pop     ax
009B  CB                 ret                     ; back to BDOS

                         page

                 ;
                 ; Function 1  Media Check
                 ;
009C             Media_Chk:
009C  EB EA              jmp     Done

                 ;
                 ; Function 2  Build BIOS Parameter Block
                 ;
009E             Build_Bpb:
009E  EB E8              jmp     Done
```

(Continued on next page)

 (Listing Continued, text begins on page 94)

```
                    ;
                    ; Function 3  I/O Control Read
                    ;
00A0                Ioctl_Inp:
00A0  EB E6                 jmp     Done

                    ;
                    ; Function 4  Read from Device
                    ;
00A2                Input:
00A2  EB E4                 jmp     Done

                    ;
                    ; Function 5  Non-destructive Read
                    ;
00A4                Nd_Input:
00A4  EB E2                 jmp     done

                    ;
                    ; Function 6  Return Input Status
                    ;
00A6                Inp_Stat:
00A6  EB E0                 jmp     Done

                    ;
                    ; Function 7  Flush Input Buffers
                    ;
00A8                Inp_Flush:
00A8  EB DE                 jmp     Done

                    ;
                    ; Function 8  Write to Device
                    ;
00AA                Output:
00AA  EB DC                 jmp     Done

                    ;
                    ; Function 9  Write with Verify
                    ;
00AC                Outp_Vfy:
00AC  EB DA                 jmp     Done

                    ;
                    ; Function 10  Return Output Status
                    ;
00AE                Outp_Stat:
00AE  EB D8                 jmp     Done

                    ;
                    ; Function 11  Flush Output Buffers
                    ;
00B0                Outp_Flush:
00B0  EB D6                 jmp     Done

                    ;
                    ; Function 12  I/O Control Write
                    ;
00B2                Ioctl_Outp:
00B2  EB D4                 jmp     Done

                    ;
                    ; Function 13  Device Open (MS-DOS 3.x)
                    ;
00B4                Dev_Open:
00B4  EB D2                 jmp     Done

                    ;
                    ; Function 14  Device Close (MS-DOS 3.x)
                    ;
00B6                Dev_Close:
00B6  EB D0                 jmp     Done

                    ;
```

(Continued on page 108)



HOW DO YOU MEASURE A C COMPILER?

- **CODE SPEED & SIZE**
  The Lattice C Compiler "generates code that is quite compact and fast running." *Peter Norton, PC Magazine*
- **CONSISTENT RELIABILITY**
  "The Lattice Compiler has performed reliably and predictably." *R. Phraner, Byte Magazine*
- **THIRD-PARTY LIBRARIES**
  More than 40 library products are currently available from Lattice
- **DEBUGGER SUPPORT**
  The Lattice C-SPRITE Debugger is now available
- **ALL MEMORY MODELS**
  Lattice C has 7 memory models available to allow the best solution for the task at hand

PLUS:
- **COMPILE TIME**
- **UNIX V COMPATIBILITY**
- **DOCUMENTATION**
- **UPDATE POLICIES**
- **COOPERATING PRODUCTS**
- **AVAILABILITY OF CROSS COMPILERS**
- **VENDOR REPUTATION**

SATISFACTION GUARANTEED!
Ask About Our "Trade Up To Lattice C" Policy

LATTICE®

Lattice®, Inc.
P. O. Box 3148
Glen Ellyn, IL 60138
(312) 858-7950
TWX 910-291-2190

International Sales Offices
Belgium: Softshop. Phone: (32) 53-664875.
England: Roundhill. Phone: (0672) 54675.
Japan: Lifeboat. Phone: (03) 293-4711.

Circle **no. 58** on reader service card.

TSF

Variable 54, Where Are You?

Is programming the lastest game of Trivial Pursuits?
- Is this the latest listing?
- Where else is this variable changed?
- Where is this procedure used?

TSF's Source Locator helps you stay productive

Source code listings
- volume, path, file name, and time-stamp
- your project and proprietary information markings
- line numbers, line-wrap at word boundries, and more

Cross-reference listings (data and procedures)
- assembler, BASIC, C, and PASCAL
- usage (value or assignment) and scope (global or local)
- your comments

Combine any number of files and languages to provide a comprehensive index for an entire program or system.

The Source Locator

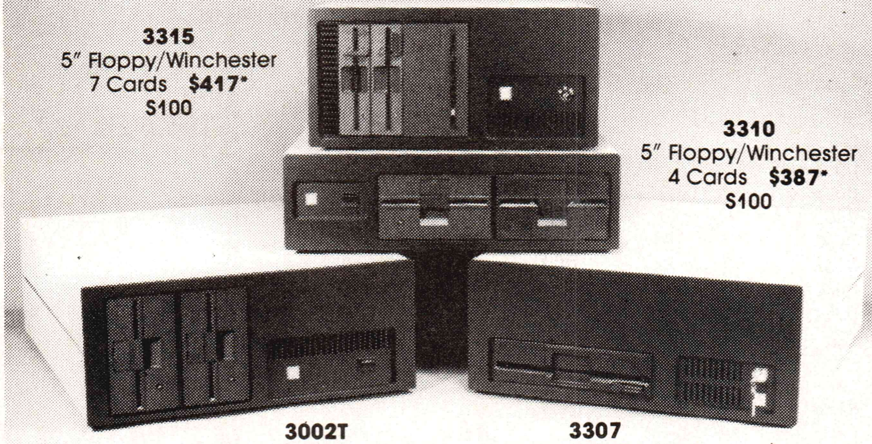
Introductory Price **$29.95**
For the IBM PC, XT, AT and compatibles, DOS 2.0+

TSF Dept. A-1
649 Mission St. San Francisco, CA. 94105
(415) 957-0111

Visa and Mastercard phone orders accepted. California residents add sales tax. Price includes shipping & handling. Dealer and Site License inquiries invited. Not copy protected.

Circle **no. 34** on reader service card.

Chassis Classics



3315
5″ Floppy/Winchester
7 Cards **$417***
S100

3310
5″ Floppy/Winchester
4 Cards **$387***
S100

3002T
5″ Floppy/Winchester
10 Cards **$565***
S100

3307
8″ Floppy/5″ Winchester
7 Cards **$494***
S100

laser 3000 **MAIN/FRAMES & DISC ENCLOSURES** FROM **$100**

LASER 3000 DISC/COVERS (not shown)

3916F
5″ Floppy **$100***

3915
2 ea. 5″ Winchester **$199***

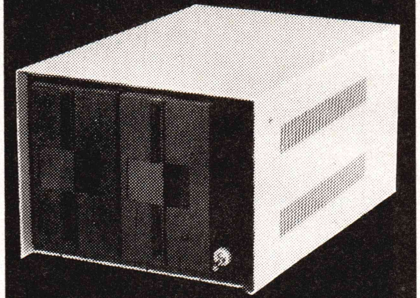
(Disk drives not included)

*1 piece; prices lower in quantity.

INTEGRAND
RESEARCH CORPORATION
8620 Roosevelt Ave./Visalia, CA 93291 209/651-1203

"Little Board" MAIN/FRAMES

6 Models from $125*



$150 (1 piece*)
MODEL 2800
Includes power supply & fan
(Disk Drives and Little Board not included)
AMPRO & Little Board are TM AMPRO computers.

INTEGRAND
RESEARCH CORPORATION
8620 Roosevelt Ave./Visalia, CA 93291
209/651-1203

Circle **no. 15** on reader service card.

```
                            ; Function 15  Removeable Media (MS-DOS 3.x)
                            ;
00B8                        Rem_Media:
00B8  EB CE                         jmp     Done
                                    page
                            ; The Initialization code for the driver is called only once
                            ; when the driver is loaded.  In this example, it returns its
                            ; own address to the DOS as the start of free memory after the
                            ; driver, so that the memory occupied by INIT will be reclaimed
                            ; after it is finished with its work.  Only MS-DOS services 01-0CH
                            ; and 30H can be called by the INIT code.
                            ;
                            ; Block device drivers must also return the number of units and
                            ; the address of the BIOS Parameter Block pointer array; if all
                            ; units are the same, all pointers can point to the same BPB.
                            ;
00BA                        Init:                           ; Function 0
                                                            ; initialize device driver
00BA  06                            push    es              ; push Request Header addr
00BB  57                            push    di
00BC  B4 09                         mov     ah,9            ; print sign-on message
00BE  BA 0016 R                     mov     dx,offset Ident
00C1  CD 21                         int     21h
00C3  5F                            pop     di              ; restore Request Header addr
00C4  07                            pop     es
                                                            ; set first usable memory addr.
00C5  26: C7 45 0E 00BA R           mov     word ptr es:[di.Address],offset Init
00CB  26: 8C 4D 10                  mov     word ptr es:[di.Address+2],cs
00CF  EB B7                         jmp     Done
00D1                        Intr    endp
00D1                        Driver  endp
00D1                        code    ends
                                    end

Structures and records:

                N a m e             Width   # fields
                                    Shift   Width   Mask    Initial

REQUEST. . . . . . . . . . . . .    0016    0009
  RLENGTH. . . . . . . . . . . .    0000
  UNIT . . . . . . . . . . . . .    0001
  COMMAND. . . . . . . . . . . .    0002
  STATUS . . . . . . . . . . . .    0003
  RESERVE. . . . . . . . . . . .    0005
  MEDIA. . . . . . . . . . . . .    000D
  ADDRESS. . . . . . . . . . . .    000E
  COUNT. . . . . . . . . . . . .    0012
  SECTOR . . . . . . . . . . . .    0014

Segments and Groups:

                N a m e             Size    Align   Combine Class

CODE . . . . . . . . . . . . . .    00D1    PARA    PUBLIC  'CODE'

Symbols:

                N a m e             Type    Value   Attr

BUILD_BPB. . . . . . . . . . . .    L NEAR  009E    CODE
CR . . . . . . . . . . . . . . .    Number  000D
DEV_CLOSE. . . . . . . . . . . .    L NEAR  00B6    CODE
DEV_OPEN . . . . . . . . . . . .    L NEAR  00B4    CODE
DISPATCH . . . . . . . . . . . .    L NEAR  0036    CODE
DONE . . . . . . . . . . . . . .    L NEAR  0088    CODE
```

```
DRIVER . . . . . . . . . . . . . .    F PROC  0000   CODE   Length =00D1
EOM. . . . . . . . . . . . . . . .    Number  0024
ERROR. . . . . . . . . . . . . . .    L NEAR  0083   CODE
EXIT . . . . . . . . . . . . . . .    L NEAR  008A   CODE
HEADER . . . . . . . . . . . . . .    L DWORD 0000   CODE
IDENT. . . . . . . . . . . . . . .    L BYTE  0016   CODE
INIT . . . . . . . . . . . . . . .    L NEAR  00BA   CODE
INPUT. . . . . . . . . . . . . . .    L NEAR  00A2   CODE
INP_FLUSH. . . . . . . . . . . . .    L NEAR  00A8   CODE
INP_STAT . . . . . . . . . . . . .    L NEAR  00A6   CODE
INTR . . . . . . . . . . . . . . .    F PROC  0061   CODE   Length =0070
IOCTL_INP. . . . . . . . . . . . .    L NEAR  00A0   CODE
IOCTL_OUTP . . . . . . . . . . . .    L NEAR  00B2   CODE
LF . . . . . . . . . . . . . . . .    Number  000A
MAX_CMD. . . . . . . . . . . . . .    Number  000F
MEDIA_CHK. . . . . . . . . . . . .    L NEAR  009C   CODE
ND_INPUT . . . . . . . . . . . . .    L NEAR  00A4   CODE
OUTPUT . . . . . . . . . . . . . .    L NEAR  00AA   CODE
OUTP_FLUSH . . . . . . . . . . . .    L NEAR  00B0   CODE
OUTP_STAT. . . . . . . . . . . . .    L NEAR  00AE   CODE
OUTP_VFY . . . . . . . . . . . . .    L NEAR  00AC   CODE
REM_MEDIA. . . . . . . . . . . . .    L NEAR  00B8   CODE
RH_PTR . . . . . . . . . . . . . .    L DWORD 0012   CODE
STRAT. . . . . . . . . . . . . . .    F PROC  0056   CODE   Length =000B
UNK_COMMAND. . . . . . . . . . . .    L NEAR  0081   CODE

49190 Bytes free

Warning Severe
Errors  Errors
0       0
```

End Listing

COMPUTER CALISTHENICS

by Michael Wiesenberg

I've received a lot of response to the first two "Computer Calisthenics" columns.

The first puzzle in the November 1984 issue was:

There are two world-famed mathematicians, Mr. P and Mr. S. A friend of theirs who likes puzzles tells both of them that he has two numbers in mind. The only thing he will tell both of them about these two numbers is that they are both integers, greater than 1, and they are not the same.

He then whispers to Mr. P the product of the two numbers.

He whispers to Mr. S the sum of the two numbers.

Mr. P knows that Mr. S knows the sum, but he doesn't know what that sum is. Similarly, Mr. S knows that Mr. P knows the product, but he doesn't know what that product is.

The following conversation then takes place:

Mr. P: "I don't know the numbers."

Mr. S: "I know that; I also don't know the numbers."

Mr. P: "Oh, then I know the numbers."

Mr. S: "Really, then I do also."

Your task, of course, is to discover the two numbers. It would take you a long time to figure them out by hand, so write a program to do it. A short *program.*

Many offered programmatic solutions that produced the correct answer, but, unfortunately, they also yielded other sets of answers that were wrong.

Let's see what's happening at each statement. I'm going to give Alan Tracht, of Cleveland Heights, OH, an honorable mention for coming up with the correct reasoning required to solve the puzzle programmatically, but he doesn't get the t-shirt because his program is one of those that yields too many answers:

(1) Mr. P: "I don't know the numbers." They are not both prime.
(2) Mr. S: "I know that ...." The numbers cannot be the sum of two primes.
(3) "... I also don't know the numbers." The sum is at least 7, so that it can be the sum of at least two different pairs of numbers.
(4) Mr. P: "Oh, then I know the numbers." Of all the factor-pairs of the product, only one sums to a number that cannot be the sum of two primes.
(5) Mr. S: "Really, then I do also." Of all the addend-pairs, only one multiplies to a product that gives Mr. P the answer (under the conditions of the previous statement).

Let's see why certain sets of numbers fail one or more of the tests.

(5, 7): P=35, S=12

Mr. P would not have made statement 1.

(5, 6): P=30, S=11

Knowing the product, Mr. P would have made his first statement. Knowing the sum, Mr. S would have made statements 2 and 3. At this point, however, Mr. P still would not know whether his product of 30 was formed from the pair (5, 6) or (2, 15), and so would not have made statement 4.

(4, 61): P=244, S=65

This was the most common incorrect answer. These two numbers do not satisfy statement 5, because (4, 61) and (8, 57) each satisfy statement 4.

Mike Meyer, of Norman, OK, provided an interesting extension of this problem, showing how this puzzle is the first of an infinite sequence of puzzles; that is, the conversation becomes:

Mr. P: "I don't know the numbers."

Mr. S: "I know that; I also don't know the numbers."

Puzzle 1:

Mr. P: "I still don't know the numbers."

Mr. S: "I still don't know the numbers, either."

Puzzle 2:

and so on. At each point, which we can label *n*, insert the sequence:

Mr. P: "Oh, then I know the numbers."

Mr. S: "Really, then I do also."

to create puzzle number *n*.

Interesting. The question is: who can *solve* one of those now?

The award goes to Charles Wells of the Department of Mathematics and Statistics, Case Western University, Cleveland, OH. The answer spit out by his program (Listing One, page 114) after running all night is (4, 13). He ran the program up to number pairs including 40, at which point he says it ran very slowly. He also implies that there might be larger answers, but "I refuse to believe anyone, mathematician or otherwise, could work this out for numbers above 20 in his head." Agreed. Perhaps I should have added to the puzzle the proviso that neither number was greater than 100, but nowhere did I say that the two world-famed mathematicians were idiot savants. Each would know that the other could not come up with an answer in his head for numbers greater than 100.

Although scores of entrants tried their hand at the preceding problem in logic, only two essayed the much easier task of programmatically describing a horse. J. C. Williams, of Cincinnati, OH, wins the prize for his program, presented in its entirety in these two lines:

```
FUNCTION HORSE( ) RETURN
    BOOLEAN
RETURN FALSE
```

He includes this "proof": "The function (program) is quite obviously a neigh-sayer," and appends the comment "(Sorry)." I presume he is apologizing for the pun, but that sort of originality was precisely what I was hoping for.

December's problem inspired more than double the entries of November's:

Assign a numerical value to each letter of the alphabet, starting with 1 for A and going up to 26 for Z. Any word in the English language has a value obtained by multiplying the values for each of its letters. For example, the word hello *is worth 86,400, obtained by multiplying 8 × 5 × 12 × 12 × 15. Which English word is equal to exactly 1,000,000? If there is none, which is the closest? Only words found in* The Random House Dictionary of the English Language *(unabridged edition) can be used. No capitalized words, none with hyphens or other embedded punctuation, nor those designated as foreign.*

Well, how about it, folks? Can you devise a program that finds the right word? And, having done that, can you tell us what that word is? Your program must be short and elegant. The algorithms can be demonstrated in a good pseudolanguage if you wish or perhaps in flowcharts.

Several entrants sent me words that multiplied out to 1,000,000, but I did not award any of them a t-shirt for several reasons. Many discovered that the word *typey* has a value of 1,000,000 and told me what dictionaries they had found it in. Others offered *tetty*, defined in the *Oxford English Dictionary* (*OED*) as a form of "testy." (It's also in my *Webster's New International Dictionary*, second edition.) The *OED* also yields *tytte*, an old form of "teat."

Those who sent me only one of these words did not qualify for the prize, because I asked for a *program*, not just an answer. Those who included a program that yielded only one of these words also did not win, because they did not follow the rules. I specified the *Random House Dictionary* for a number of reasons, and it does not contain *typey, tetty,* or *tytte*.

Why this particular dictionary? It is available as a dictionary program and thus does not necessitate typing in a whole dictionary full of words. Also, it is free of what I call "crossword puzzle dictionary words"—obsolete words, archaic variants, and other words that are seldom found in use today. Furthermore, *not all the words in it are in the electronic dictionary*. No currently available electronic dictionary even comes close to having as many words as its "hard copy" counterpart does. It is an unusual spelling checker that has even 50,000 words, and yet the tome of over 2000 pages that is my main reference work contains in excess of 260,000 *entries*, and there are many more words than entries. For example, *tablet* is a separate entry, but *tableted* is not. *Kaleidoscopically* is not a separate entry; it is found at the end of the definition for *kaleidoscope*. Spelling checkers are even more limited: they usually have only one form of each word. The plurals of nouns, the progressive forms of verbs, and so on, are often not there. The point I wished to make, which most people did not get, was that perhaps brute force, in the form of examining every word in a given dictionary, is not the way to solve this puzzle.

I commend all those who wrote programs generating all possible candidates and embedded in their programs the logic that the letters forming the hypothetical word must be multiples of 2 or 5 or both, those being the only prime factors of 1,000,000; that is, if a word exists in the specified dictionary that multiplies out exactly to 1,000,000, it must consist of some combination of B, D, E, H, J, P, T, and Y. Any qualifying word can have,

in addition, any number of As, that being a "free" letter because its addition to a word merely multiplies the product by 1. In fact, Ulrich Sondermann, of Concord, CA, chastised me for specifying the values of the letters to be 1 through 26, rather than 2 through 27—which, he said, was the way the problem was originally stated in the puzzle magazine. (The best word in that contest was *ixodid*, a kind of tick; Ulrich submitted the correct answer in that contest.) Ah, but I did not wish to repeat a previously published puzzle, and I *did* want a "free" letter because I wanted to provide a different kind of puzzle.

Ulrich deserves mention for another reason. He wrote his program in BASIC on a Timex/Sinclair: quite a feat! It generated 123 candidates (using the values I specified) in four minutes. He also provided a good algorithm for finding out if a given electronic dictionary contains any words consisting of these letter combinations. He said, "The problem then consists of creating a dictionary with words that have letters sorted in descending order. A match can then be made with each of the 123 candidates." (But what about the 124th candidate, Ulrich?)

I commend those of you who wrote such programs, but I did not award you prizes because I am not yet convinced that the *Random House Dictionary* has a word multiplying out to exactly 1,000,000 and because, with one notable exception, none of the programs accounted for the fact that adding any number of As to a word does not change its value.

Among the preceding, I give honorable mention to both Ken Waldron, of Vancouver, BC, and Ray Gardner, of Englewood, CO, because each devised a program that produced the same 124 candidates (all sans As). I also include Ray's program (Listing Two, page 116), because he accomplished in 24 lines of C code (seven lines of which, in good C coding practice, consist merely of one brace) what took Ken a page and a half of Pascal. Ray claims his program, written in Lattice C, runs in under four seconds on his IBM PC. Ray also says that his wife found the word *typey* in 15 minutes, without the benefit of a computer. (So much for computers.)

I also mention Thomas Shores, of Lincoln, NB, because, while his C program does not specifically generate words with As, he did acknowledge in his accompanying letter that they should be inserted "appropriately" within generated candidates. Also, his program generates all possible candidates, not just those multiplying out to 1,000,000. The best word he found was *moors*, value 1,000,350, but he thought others might come up with better possibilities. He stated that if any better words were found, they would have one of these values: 999,702; 999,810; 999,856; 1,000,000; 1,000,188. Good figuring, particularly since the word *I* think is the best has a value of 1,000,188. (How come you missed *rooms*, Thomas, an anagram of *moors*, and *comers*, which is also value 1,000,350?)

Trent Garverick, of Columbus, OH, gets an honorable mention for his short and sweet self-documenting pseudocode algorithm, the implementation of which in UCSD Pascal on an Apple II yielded what I believe to be the word *closest* to 1,000,000 found in the specified dictionary. *Curing* multiplies out to 1,000,188.

My own program had come up with that very word. My program first generated candidates within plus or minus 1000 of 1,000,000. I then "eyeballed" the list for real words. When I found several within 350, I narrowed the variance. Each time I found a better word, I started again with the smaller offset. My program, however, was inefficient, it took forever to run, and it generated literally thousands of possibilities, mostly because it spit out all permutations of each "word," not just words sorted in alphabetic descending order.

Trent based his program on examination of a real dictionary, rather than on generation of candidates. I still don't think that is the best method, but his program at least does not assume that a word multiplying out to exactly 1,000,000 must exist nor does it refuse to account for words with As in them.

I also mention Tom Balon, of Apalachin, NY, and Bob Smith, of Windsor, NY, who jointly produced a Pascal program on a VAX 11-785 to examine a dictionary of 33,595 words, coming up with *curing*. They also found, tied at 1,000,188, *Nicaragua*, but that word is disqualified as being a proper noun. Their program, while relatively short, was not self-documenting. Joe Celko, of Los Angeles, also deserves mention for a short, relatively easy-to-follow C program that produces, in combination with a spelling checker, lots of possibilities, the best of which he concludes is *curing*.

And now for the winner. The envelope, Randy, please. Ah, yes, Fred Smith, of Stoneham, MA, wrote a short C program into which you input a list of words; the program then determines the value of each word. The

```
goatees     997500
soybean     997500
stogy       997500
suety       997500
alkaline    997920
budlike     997920
chuckle     997920
clinked     997920
driven      997920
flanker     997920
invader     997920
lurked      997920
pucker      997920
variant     997920
viaduct     997920
village     997920
mopped      998400
palmated    998400
banquet     999600
curing      1000188
nicaragua   1000188
comers      1000350
cozies      1000350
moors       1000350
rooms       1000350
seamier     1000350
arkansas    1000692
sulks       1000692
packets     1003200
shakeable   1003200
shelve      1003200
shoved      1003200
specked     1003200
stalked     1003200
```

Figure
Partial Listing of Words
Produced by Fred Smith's Million.C

default is to print only those words whose values lie within 10,000 of 1,000,000, but it has a command line switch (great for debugging) that prints all values. The program also ignores words of less than five letters. And here is the clever part, the spark of originality for which Fred really deserves the t-shirt: he then used the powerful capabilities of Unix to examine an entire dictionary, one of over 90,000 words.

He used two Unix tools, each performing one task well. (One was his own word value-determination program.) His program, *million*, reads all the words in a 90,000-word dictionary called *words* and pipes its output to the *sort* utility, which performs a numeric sort on the second field of each line and writes the final sorted file to an output file called *m.w.sorted*. Fred's program is Listing Three (page118). Here is the Unix command:

```
million < words sort -n +1
  > m.w.sorted
```

So why, if I'm not convinced that reading all the words of a dictionary—furthermore, one that does not contain all the possible words of the dictionary I specified—is the best method, do I award Fred the t-shirt? Two reasons. One, his program was the cleverest. Two, nobody did it the way I thought it should be done.

I was hoping that someone would devise a program that generates all candidates equal to and close to 1,000,000 and retains only those obviously or likely to be English words, coming up with a small list that *a human being* then could scan for "real" words. What I wanted to show was the necessary cooperation between machines and humans on problems of this sort.

Russ Nelson, of Potsdam, NY, also deserves mention because he was clever enough to figure out precisely at which sprawling Silicon Valley corporate octopus my curious collection of I-Q Industries eccentrics *actually* works, proving to me that his knowledge was no mere guess by replicating the convoluted programmatic word game with which I produced the name. I would have awarded him an honorary t-shirt if he had also figured out what classical bit of entertainment inspired the transposition.

Thanks for all the kind letters!

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 197**.

Computer Calisthenics (Text begins on page 110)

Listing One

```
PROGRAM DAVIS(INPUT,OUTPUT);

CONST LARGE=50;

VAR M,N:INTEGER;

(*AT EACH STAGE BELOW, N REFERS TO THE
  NUMBER THAT MATHEMATICIAN WAS TOLD.*)


FUNCTION STAGEA(N:INTEGER):BOOLEAN;

  (*AT STAGE A, MR. P DOESN'T KNOW.
    THAT MEANS N HAS MORE THAN ONE
    ACCEPTABLE FACTORIZATION.*)

    VAR COUNT,F,G:INTEGER;

      BEGIN
        F:=2; G:=N DIV 2; COUNT:=0;
        WHILE F<G DO
          BEGIN
            IF N MOD F = 0 THEN
              COUNT:=COUNT+1;
            F:=F+1; G:=N DIV F
          END;
        STAGEA:=(COUNT>1)
      END;


FUNCTION STAGEB(N:INTEGER):BOOLEAN;

  (*AT STAGE B, MR. S KNOWS MR P. DOESN'T
  KNOW.  THAT MEANS EACH ACCEPTABLE PARTITION
  S+T=N YIELDS A NUMBER S*T WHICH SATISFIES
  STAGE A.  HE ALSO DOES NOT KNOW HIMSELF,
  WHICH MEANS N HAS MORE THAN ONE ACCEPTABLE
  PARTITION, WHICH MERELY MEANS N>6.*)

    VAR S,T,COUNT:INTEGER;
    TEMP:BOOLEAN;

    BEGIN
      S:=2; T:=N-S; COUNT:=0; TEMP:=(N>6);
      WHILE S<T DO
        BEGIN
          COUNT:=COUNT+1;
          TEMP:=TEMP AND STAGEA(S*T);
          S:=S+1; T:=N-S
        END;
      STAGEB:=(COUNT>1) AND TEMP
    END;


FUNCTION  STAGEC(N:INTEGER):BOOLEAN;

  VAR F,G,COUNT:INTEGER;

  (*AT STAGE C, P KNOWS.  THIS MEANS THERE
  IS EXACTLY ONE ACCEPTABLE FACTORIZATION
  OF M*N FOR WHICH STAGE B IS CORRECT*)

      BEGIN
        F:=2; G:=N DIV F; COUNT:=0;
        WHILE F<G DO
          BEGIN
            IF (N MOD F = 0) AND STAGEB(F+G)
              THEN COUNT:=COUNT+1;
            F:=F+1; G:=N DIV F
          END;
        STAGEC:=(COUNT=1)
      END;
```

(Continued on page 116)

DDJ Classifieds

Software

DriveLiner

Drive Alignment Test Program for CP/M 2.2/3.1 With Dysan 8" SSSD Diagnostic Disk $65 check/MO postpaid Chandler Software, 273 W Shore Dr, Marblehead MA 01945 (617) 631-4685

NATIONAL PUBLIC DOMAIN

Why reinvent the wheel?
Public Domain Software isn't copyrighted, so no need to pay license fees for thousands of routines; business-utilities-dbase. Complete, latest issue user group disks available to rent 'n copy at your leisure. Send $5.00 for a postpaid directory disk or a SASE to National Public Domain Software, 1533 Avohill Dr., Vista, CA 92083 or (619) 727-1015.

TECMAR GRAPHICS LIBRARY

TEKMAR lets you do high-res graphics on your TECMAR Graphics Master. Features windowing, viewporting, clipping, axis rotation. Similar to Tektronix graphics. Includes screen dump/restore, Epson screen print, support for HP and Western Graphtec plotters. Includes three curve-fitting programs and graphics application SOURCE CODE. Requires MS-FORT 3.20, or Lahey F77L. Price: $195.
ADVANCED SYSTEMS CONSULTANTS,
18653 Ventura Boulevard, Suite 351,
Tarzana, California 91356
(818) 990-4942

FED "Binary File Editor"
Allows the user direct access in both HEX and ASCII format to any disk file on the IBM PC. FED can patch object modules, examine word processing files, repair damaged files and verify the results of I/O operations. S/S DOS disk for 128 IBM PC. Only $49 "+ $5 s/h".
The Whitewater Group
2912 N. Burling Avenue
Chicago, IL 60657
(312) 975-6095

W.B. SOFTWARE DEVELOPMENT

BROWSE for CP/M-68K
for looking at any CP/M-68K file
scroll up, down, left, right
powerful FIND (search) command
online HELP, PFKEYS, TABS, PRINT
100% 68000 assembler—ASCII, HEX
ANSI 3.64 & other terminal types
$49.95 US. Visa, M/C, money order.
WB Software Development,
112-Oakhampton Pl. SW,
Calgary, AB,
Canada T2V 4B2 (403) 238-3216

ECLIPSE SYSTEMS

AZTEC C65 + ProDOS

Use Aztec C DOS 3.3 software under ProDOS. Run programs without relinking. System includes recursive Shell, support for pseudo-code, vi like editor with macros, library upgrade and source code. Requires ProDOS user's disk. $49.95.
Eclipse Systems, 223 Matthew Rd., Marion, PA 19066. (215) 667-8354

Utility

MICRO COMPUTING SERVICES

CBTREE FOR C PROGRAMMERS
Provides enhanced file handling calls directly into C programs. Maintains balanced B-trees, supports unlimited number of keys and key lengths. Fast, Flexible, Efficient. No royalties. Source Code Incl. $179. MICRO COMPUTING SERVICES 2009 Hileman Road Falls Church, VA 22043 (703) 893-0118

Publishing

OWINGS MILLS

OHIO SCIENTIFIC/ISOTRON AND COMPATIBLE COMPUTERS! Users MUST read PEEK (65). Published monthly exclusively for above users. $19/yr U.S. $26/yr Canada; P.O. Box 347, Owings Mills, MD 21117; (301) 363-3268.

SOFTWARE SENTINEL
A hardware key that prohibits unauthorized use. Its benefits: Unlimited backup copies; unbreakable; site licensing control; no floppy required with hard disk; transportable; transparent; pocketsize. Evaluation Kit available. PC compatible.
Rainbow Technologies, Inc.
17971 Skypark Circle, Suite E
Irvine, CA 92714 (714) 261-0228

DIAL & ORDER
24 HR. MODEM LINE
(408) 425-1371
Specializing in technical/reference books and computer supplies.
Wise Dog Computerbooks
1310 Fair Ave, Santa Cruz, CA 95060
Toll Free N. Cal. (800) 558-9473

CONTROL SYSTEMS SOFTWARE ENGINEER

A Challenging Opportunity in our Florida Corporate Offices

RS&H offers a challenging position with responsibility for development of systems level and applications level software for various Industrial Process Control applications.

Responsibilities will include involvement in associated hardware, and control strategy design, but a strong systems software background is the primary requirement.

Candidates should have at least three years' experience and a strong working knowledge of at least one computer operating system, assembly language, and "C". Familiarity with various DEC equipment operating systems, "C" and MACRO is particularly desirable. Experience with development of real-time software with some Process Control/Instrumentation background is desired. A BS degree in Computer Science or Engineering is a minimum requirement.

RS&H is a multi-disciplined A/E/P firm with a 43 year reputation for innovative technology applications and professional development. **RS&H** offers an attractive salary and benefits package including a prime Florida location providing a quality of life second to none. Please send your resume in complete confidence, including salary history to **Thomas M. Hills (904) 739-2000.**

REYNOLDS, SMITH AND HILLS
Architects-Engineers-Planners, Incorporated
P.O. Box 4850
Jacksonville, FL 32201

RS&H encourages qualified minorities, women, veterans and handicapped individuals to apply

EOE/AAP

Dr. Dobb's Journal
is pleased to announce the
DDJ Classifieds

RATES: DISPLAY ADVERTISERS: Price per column inch $100. Ad must run in 3 consecutive issues.
LINE ADVERTISERS: Price per line $12 (35 characters per line including spaces). Minimum of 5 lines. 3 consecutive issues. Add $3 per line for boldface type. Add $25 if a background screen is desired.

DISCOUNTS: Frequency discounts for 6 consecutive ads (less 7%) and for 12 consecutive ads (less 15%).

MECHANICAL REQUIREMENTS: Camera ready art or typewritten copy only (phone orders excepted). Display: Specify desired size, include $20 if reduction is required. Line: Specify characters in boldface, caps. Allow 6 weeks for publication.

PAYMENTS: Full payment in advance. Check, money order, Visa, M/C, AmEx.

Run this ad in ______ issues
Size: ______ col x ______ inches or 1 col x ______ lines
Payment by: ______ check ______ m/o ______ Visa ______ M/C ______ AmEx
Card # ______________________ Exp Date ______
Signature ______________________
Print Name ______________________
Company ______________________
Address ______________________
City ______________ St ______________ Zip ______
Phone ______________ Current Subscriber ______

Mail to or phone Tim Ortiz, DDJ Classifieds, 2464 Embarcadero Way, Palo Alto, CA 94303 (415) 424-0600

Listing One

```
FUNCTION STAGED(N:INTEGER):BOOLEAN;

  VAR S,T,COUNT:INTEGER;

  (*AT STAGE D, S KNOWS TOO. THIS MEANS
    THAT THERE IS EXACTLY ONE ACCEPTABLE
    PARTITION S+T OF N FOR WHICH STAGE
    C IS CORRECT FOR S*T.*)

      BEGIN
        S:=2; T:=N-S; COUNT:=0;
        WHILE S<T DO
          BEGIN
            IF STAGEC(S*T) THEN COUNT:=COUNT+1;
            S:=S+1; T:=N-S
          END;
        STAGED:=(COUNT=1)
      END;

BEGIN (*MAIN*)

FOR M:=3 TO LARGE DO
  BEGIN
    WRITE(M:3);
    IF (M MOD 12 = 0) THEN WRITELN;
    FOR N:= 2 TO M-1 DO
      IF STAGEA(M*N) THEN
        IF STAGEB(M+N) THEN
          IF STAGEC(M*N) THEN
            IF STAGED(M+N) THEN
              BEGIN
                WRITELN;
                WRITELN('THIS PAIR WORKS:',N:3,M:3);
              END;
  END;
END.
```

End Listing One

Listing Two

```
/*      solve   --  Computer Calisthenics solution
*/

char letter [] = "bdehjpty";
char solution[20];

main ()
{
    solve(0,0,1000000L);
}

solve (length, nextletter, goal)
int length, nextletter;
long goal;
{
    int i, value;
    if ( goal == 1 ) {                          /* found a solution, write it  */
        solution[length] = '\0';
        puts(solution);
    } else {                                    /* partial solution, extend it */
        for ( i = nextletter; letter[i]; i++ ) {
            value = letter[i] - 'a' + 1;
            if ( goal % value == 0 ) {
                solution[length] = letter[i];
                solve(length+1, i, goal/value);
            }
        }
    }
}
```

End Listing Two

(Continued on page 118)

DSD 80

FULL SCREEN SYMBOLIC DEBUGGER

"THE SINGLE BEST DEBUGGER FOR CP/M-80. A TRULY AMAZING PRODUCT."

LEOR ZOLMAN
AUTHOR OF BDS C

- Complete upward compatibility with DDT
- Simultaneous instruction, register, stack & memory displays
- Software In-Circuit-Emulator provides write protected memory, execute only code and stack protection.
- Full Z80 support with Intel or Zilog Mnemonics
- Thirty day money back guarantee
- On-line help & 50 page user manual

NOW ONLY $125.

SOFTADVANCES

P.O. BOX 49473 AUSTIN, TEXAS 78765 (512) 478-4763

MasterCard VISA

Circle **no. 63** on reader service card.

A FULL C COMPILER FOR $49.95

The Ecosoft Eco-C88 compiler for the 8088 and MSDOS is going to set a new standard for price and performance. Consider the evidence:

Compiler	Eco-C88	Latice (1)	C86 (1)
Seive	13	11	13
Fib	44	58	46
Deref	13	13	–
Matrix	21	29	27
Price	$49.95	$500.00	$395.00

(1) **Computer Language,** Feb., 1985, pp.73-102. Reprinted by permission.

The Eco-C88 compiler is a full K&R C compiler that supports all data types and operators (except bit fields). Now look at the other features we offer:

- ⋆ 8087 co-processor support using a single library. If you install an 8087 later, the software will use it without having to recompile.
- ⋆ A robust standard library with over 150 functions, including transcendentals, color, and others.
- ⋆ OBJ output for linking with the MSDOS linker (LINK).
- ⋆ Error messages in English – no cryptic numbers to look up. A real plus especially if you're just getting started with C.
- ⋆ Easy-to-read and complete user's manual.
- ⋆ Works with all IBM and compatibles running MSDOS 2.0 (or later).
- ⋆ Plus many other features.

For $10.00 more, we will include the source code for the C library functions (excluding transcendentals). For an additional $15.00, we will include our ISAM file handler in OBJ format (as published in the **C Programmer's Library,** Que Publishing). The discount prices for the library source and ISAM only apply at the time the compiler is purchased. Please add $4.00 to cover postage and handling. To order, call or write:

Ecosoft Inc.
6413 N. College Avenue
Indianapolis, IN 46220
(317) 255-6476

MasterCard VISA AMERICAN EXPRESS

ECOSOFT INC.

Eco-C (Ecosoft), MSDOS (Microsoft), UNIX (Bell Labs), CP/M (Digital Research), Z80 (Zilog), 8086, 8087, 8088 (Intel).

Circle **no. 35** on reader service card.

AT LAST

S-100 ⟺ 488

THAT DOES EVERYTHING YOU WANT IT TO DO

DWD

D&W DIGITAL, INC.
20655 Hathaway Avenue
Hayward, California 94541
(415) 887-5711

Circle **no. 28** on reader service card.

Listing Three

```
            c  /*
            c  ---------------------------------> million.c <---------------------------------
            c
            c  Usage: million [-nl] < infile > outfile
            c
            c  I/O is from stdio. By default, only words with products >= 990000 and
            c          <= 1010000 are output. If -nl switch is used ('no limits'), the
            c          limit test is suppressed, and all words >= 5 letters in length are
            c          printed.
            c
            c  */




               #include <stdio.h>
               #include <ctype.h>

               #define OPT     1000000                              /* optimum value        */
               #define MIN     OPT - 10000                          /* minimum acceptable   */
               #define MAX     OPT + 10000                          /* maximum acceptable   */
               #define BOOL    unsigned short
               #define MAXLINE 80                 /* longest permissible input string       */
               #define TRUE    1
               #define FALSE   0

               main(argc,argv)
                       int argc;
                       char *argv[];
 0-----------          {
 |                     char word[MAXLINE];          /* input word to be examined              */
 |                     register char *pw;           /* register pointer thereto               */
 |                     register int wordlen;        /* length of string stored in word        */
 |                     register unsigned long prod; /* working word value                     */
 |                     register BOOL print = TRUE;  /* ok to print word if TRUE               */
 |                     register BOOL limit = TRUE;  /* assume MIN-MAX range limits            */
 |
 |                     if ((argc > 1) && !strcmp(argv[1], "-nl"))
 |1----------              {                        /*  suppress limit test for printing     */
 ||                        limit = FALSE;
 |1----------              }
 |                     while ( gets(word))                     /* until end of input, get next word */
 |1----------              {
 ||                        if ((wordlen = strlen(word)) < 5)   /* filter short words */
 ||                            continue;
 ||                        for (print = TRUE, prod = 1, pw = word ; *pw ; pw++)
 ||2--------c                  {                    /* examine for non-alpha characters,
 |||        c                                       calculate product                         */
 |||                           if ( ! isalpha(*pw))
 |||3-------                       {
 ||||                              print = FALSE;
 ||||                              break;
 |||3-------                       }
 |||                           prod *= (tolower(*pw) - 'a' + 1);
 ||2--------                   }
 ||                        if (((prod < MIN) || (prod > MAX)) && limit)
 ||2--------                   {                               /* check limit conditions         */
 |||                           print = FALSE;
 ||2--------                   }
 ||                        if (print)
 ||2--------                   {                               /* is it ok to print? */
 |||                           printf("%s",word);                     /* print word     */
 |||                           for ( ; wordlen < 15 ; wordlen++)
 |||3-------                       {                           /* pad with blanks        */
 ||||                              putchar(' ');
 |||3-------                       }
 |||                           printf("%u\n",prod);            /* print word value       */
 ||2--------                   }
 |1----------              }
 0-----------          }
```

End Listings

$49.95 **FMT** **$49.95**

Text Formatting System

FMT provides most of the features of the high-priced Text Formatters at our inexpensive price -- and it's easier to use, too! Note the features below:

- Easily configured to your printer. Configuration files for 20 + printer models are provided or generate your own.
- FMT gets the most from your printers by taking advantage of their special features, including condensed, double width, enhanced, double print, italics, elite, letter quality, multiple fonts, etc.
- Multiple modes and combinations of modes can be used on the same line or even in the same word.
- FMT works with your favorite editor!
- FMT uses meaningful mneumonic commands in the style of SCRIPT or ROFF (each command appears on its own input line), including commands for the various printing modes.
- No embedded control codes - you don't have to remember those strange escape/control sequences.
- FMT runs at the maximum speed your printer allows for each printing mode - graphics mode is not required.
- Standard formatting features provided, including headers and footers, automatic page numbering, text justification, tabs for table generation, and embedded files up to TEN deep.
- FMT automatically builds Table of Contents, List of Figures, and three level alphabetized Index.
- Detailed 100 + page manual profusely illustrated with examples.
- Works equally well with IBM-PC, TI-PC, IBM clones and look-alikes (PC-DOS/MS-DOS 128k). Also works with CP/M 8080 and Z80 systems with 64k.
- $49.95 plus $2.00 shipping and handling.

Specify system.

VISA and Master Card Accepted

Dealer Inquiries Welcome

TINY TEK, INC.
Route 1, Box 795
Quinlan, Texas 75474
(214) 447-3025

Circle **no. 39** on reader service card.

Periscope ... A Direct Hit!

"A great debugger ... If you've been frustrated by programs that don't behave the same for the debugger as when running alone, or that crash altogether, you should check out Periscope ..."

PC REPORT, Boston Computer Society

"It works, and works well!! In the first day of use I finished up two weeks of problems!! Really excellent!!!"

Peter Loats, Periscope User

Periscope's differences hit home! Its breakout switch gives you control, even when interrupts are disabled. Periscope's unique power helps you solve the difficult problems. Debug device-drivers, memory-resident, and non-DOS programs.

Great for straightening out confused C pointers! Using its breakpoint power, you can stop on reads and writes to ranges of memory. Stop on registers, words, and bytes, too.

It's tough. Periscope's 16K RAM board is write-protected. Runaway programs can't touch crucial debugger code! Examine the system, safely, at any time: Periscope saves the interrupt vectors and buffers it uses, then restores them when done. It uses ROM BIOS calls for functions except file access, so DOS can't clobber itself.

It gets you moving. It's commands are similar to Debug's, so you'll master it quickly. On-line help is there when you need it. You can define the windows you want; you can design easy-to-read memory displays. Periscope supports C, Assembler, BASIC, and Pascal. Comprehensive symbol support gives you a roadmap through memory tying back to your source.

It's risk-free. Periscope is backed up by a 30-day, money-back guarantee! The board is warranted for a year. Technical Support? You get the best there is ... direct from Brett Salter, Periscope's author!

It requires. An IBM PC, XT, AT, Compaq, or close compatible; PC-DOS, 64K RAM; a disk drive and an 80-column monitor. With two monitors, Periscope's screen is displayed on the monitor not in use. With one monitor, a keystroke switches screens.

It's power you can afford. At $295, you can afford Periscope's professional power. The system includes the memory board, breakout switch, debugger software, manual, and quick reference card.

Order now! Call toll-free (800) 722-7006 to place your order or to get more information. MasterCard/VISA accepted.

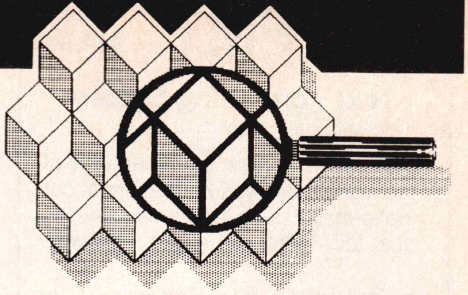
Get your programs up and running;

UP PERISCOPE!

Data Base Decisions/14 Bonnie Lane | Atlanta, GA 30328/(404) 256-3860

Circle **no. 31** on reader service card.

OF INTEREST



by Michael Swaine

The marketing of microcomputer technology is not an exact science. Lee Felsenstein, designer of the Osborne 1 and a founder of the Community Memory Project that wrote the **Sequitur** relational database management system, is now selling Sequitur through his company, Golemics, at a third of its original price. Sequitur was generally regarded as hot stuff when introduced some three years ago, but CM's faulty marketing kept it off the dealers' shelves, an article of furniture to which Sequitur will remain a stranger, since Sequitur will now be sold strictly by mail order at $250. "I kept my mouth shut [earlier] because I thought I didn't know enough to criticize. My Osborne experience made me realize that nobody else knows any better either," Lee says. He's wiser now; he turned forty this spring . . . Micro Craft, manufacturer of the **Dimension** computer, a pricey multiprocessor machine curiously positioned on release as a do-all end-user product, has also recently risen from the rubble of bad marketing. The company is now selling dual-68000 Unix machines and enjoying post-Chapter 11 prosperity . . . New Functions, the distributor of **Judy**, a concurrent personal assistant (aka gal/man Friday, sidekick, right-hand man, desktop accessory or pop-up), has one of the more novel distribution schemes extant. Judy owners can generate and give away Judy "clones"—limited-time free samples—and when the sampler buys his own Judy, the clonemaker gets a commission. New Functions recently raised that commission from 10% to 20%.

The largest computer show in the world is not held in the U.S. Atari introduced its 68000-based **ST** computer at the mammoth Hannover Fair in Germany in April. Despite the fact that the ST had a tendency to crash when asked to do anything much beyond the prepared demo, our correspondent on the scene found both the machine and its reception impressive, and advised us to look into software development for the ST under the GEM environment . . . Jack Tramiel's former company, Commodore, which has long stood strong in Europe, lost money this spring for the first time ever as dealers held out for the new **128** machine . . . A new **European Forth chip** could provide some competition for the **Novix NC4000 Forth chip**, although the Novix chip should actually be available by the time you read this. The NC4000 was designed by Charles Moore, who created Forth, and Robert Murphy; it can address four megabytes of memory and is intended to execute Forth code on the order of ten times the speed of any software implementation of Forth. Its designers expect it to find application in speed-critical areas like animation.

Boards

RAM price wars are behind price reductions on some hardware enhancements. The price of a do-it-yourself Macintosh upgrade is now below $200 . . . DFE has cut prices on its **NS32032 coprocessor board** for IBM PC/XT/AT and PC-compatible computers. The basic Tiger-32 was introduced last summer with a 6MHz clock, 32082 MMU, 512K onboard RAM and Xenix 3.0 for $2495, but RAM price-cutting has brought it down to $2095 . . . Univation's **Turbocharger** is a 9.54 MHz 8086 board for the IBM PC that sells for $1295 with 640K on-board RAM . . . Real Time Devices is selling a **GPIB instrumentation bus interface board** for PCs for $299, a per-channel cost of about $20.

Some new boards refine existing capabilities: Interstate Voice Products announced a **speaker-dependent IBM PC board** that has some limited facility for recognizing words in connected speech . . . We're examining STB's **video board**, which runs all IBM-compatible software, whether it was designed for a color or a monochrome monitor, by converting graphics displays into full-screen 16-level grey-scale images . . . ANEX Technology is selling a **banked 2-megabyte RAM** board for PCs that are running multi-user software.

For $61 (quantity one price), you can pick up a VLSI chip that lets you directly replace an 8088 microprocessor with an 80286. Edsun Labs' **EL286-88 Processor Converter** stands between the 286 and the peripherals, playing an 8088 to the peripherals, while accepting and converting all 16-bit transfer requests from the 286. The idea is to combine 16-bit main memory operations with low-cost 8-bit peripherals.

MSDOS

Alternatives or extensions to MSDOS/PCDOS are popping up like pop-ups. DRI has announced the first software to run under its Graphics Environment Manager (GEM): **GEM Desktop, Draw, Write, Paint, Wordchart and Graph**. Desktop lets you run up to six utility programs on top of applications, like the Mac's desktop accessories; all are supposed to be shipping as you read this . . . Digitrol's **pcShare** substitutes a multi-user system with file protection for PCDOS . . . RTCS's **PC/RTX** substitutes a real-time O/S (an implementation of Intel's iRMX-86) . . . Soft-

Logic's **DoubleDOS** makes PCDOS multitask for $99 ("multi" here means "two").

For software development under MSDOS/PCDOS, Phoenix software has a program development manager called **Pmaker** that they say is similar to the Unix "make" command, and a program performance analyzer caller **Pfinish** . . . **Advance Trace86** from Morgan Computing is a debugger/ assembler that lets you back up execution 20 instructions . . . Genesis's **GeneScope** is a new full-screen symbolic debugger for the PC.

Whether developing expert systems on a micro is the Next Big Thing or just the latest fad, it's getting cheaper and easier. **Insight 2** from Level Five Research lets you incorporate Pascal routines and dBase II files into a knowledge base developed on a PC or compatible . . . AI pioneer Donald Michie's **Expert Ease** package has undergone some changes of ownership and distribution (it's now distributed by Human Edge), the most visible consequence of which is a price cut from $2000 to $695.

C

"To boldly go where no C software distributor has gone before . . . ." Phoenix calls **Pre-C**, its lint superset, the industry's first program analyzer for the C language designed to support MSDOS and PCDOS . . . Micro Software Developers says its **C Debugger** is the first true source debugger for C . . . Catalytics has released what it calls the first complete C interpreter, its **Safe C** . . . For Lattice C programmers who don't necessarily hanker to be first, or at least would like some company out there on the giddy edge, Bill Hunt, author of *The C Toolbox*, has started the **Lattice C User's Group**.

Whitesmiths has upgraded its 8086-family **C compiler** to support all 8086 memory models and for closer adherence to their best guess at the emerging ANSI standard . . . Toolworks has moved its **C/80 compiler** to MSDOS and is selling it for $49.95 . . . Smith & Smith Associates has released **CrossRefC**, a C cross-reference/listing generator, for $39.95 . . . Vance Info Systems announced **C Lib**, a C function library of over 200 routines, for the MSDOS environment.

CP/M

CP/M users needn't feel left out of the pop-up fad: Poor Person Software's **Write-Hand-Man** (that's something like a sidekick, we suppose) includes a notebook, phonepad, desk calendar, file and directory viewing programs and a communications program; new functions can be added fairly easily. It sells for $49.95 and runs under CP/M 2.2 on all CP/M machines.

Also for CP/M, **MB+ Tools** is Minnow Bear Software's $175 set of programmer productivity tools for Pascal MT+ . . . Soft Advances says that its **DSD80 full-screen symbolic debugger** is fully DDT-compatible and fully supports Z80 instructions using either extended Intel or Zilog mnemonics . . . Selfware's **Convert** 3.1 lets you read, write and format disks, copy files and list directories in 105 CP/M formats on an MSDOS machine, and costs $99.

Mac

Just now, despite such attractions as the novelty of a Smalltalklike Forth

extension (Kriya's **Neon**), some of the most interesting products for the Mac are to be found on user group disks. Not surprisingly, a high proportion of these public-domain or user-supported programs consists of communications software, like Dennis Brothers' **MacTEP** and Scott Watson's **Red Ryder**. You can pick more fruit if you have a ladder . . . There are also a lot of fonts, desk accessories and graphics; the **IM Underground's graphics** consist of schematics for the machine . . . Equally unsurprising, given the features of the Mac, is the number of attractively-designed Mac newsletters, including **MacTutor**, **The Mac Street Journal** and **MacBriefs**.

Apple][

Contrary to the impression given at its stockholders' meeting early this year, Apple makes another computer, the Apple bracket-bracket. Programmers are even continuing to write software for the bracket-bracket . . . MicroSparc has released the ProDOS version of its **Ampersoft** utility package at $49.95 for Applesoft programmers . . . There is, of course, much existing Apple][software: **InCider** is selling disks of software from the magazine at $21.47 . . . **Pandora Software** has collected a library of 4000+ public-domain programs for the Apple][and is selling them at $5 per disk (at twenty programs per disk, that's $1000 for the library, but still).

A Visit to the Fab Lab

A blip of *deja vu* struck as I read in the March *IEEE Software* about Ivan Guzman de Rojas and his plan for doing natural-language translation via the peculiarly algebraic Andean Indian language Aymara. Yes, trivial recall was working perfectly: halfway down the Andean peak of press releases on my desk I found the announcement of the opening of the **Aymara Fab Lab** in Sunnyvale. A coincidence worthy of a drive down the peninsula.

Dean Norman, Director of the Aymara Fab Lab, greeted me at the door and began talking immediately about Guzman. Norman explained that Guzman, discovering that Aymara, lacking irregular verbs and gender, was an unprecedentedly logical language (although its logic was not standard two-valued logic), had succeeded in codifying the algorithmic structure of its syntax. For the first time, someone had expressed a natural language in software. Wasn't Guzman, I asked, considering the application of his achievement in the design of translating machines, the notion being that computerized Aymara could serve as the bridge in a multilanguage translation system?

Right, Norman answered, although at AFL they were taking the process in a somewhat different direction. Did I recall the Sapir-Whorf hypothesis from linguistics? I did, more or less: that language delimited the thinkable thoughts, and thus our culture and perceptions. Under normal conditions, we see only those distinctions for which we have words. In cultures in which green is not a major linguistic division of the spectrum, it is also not a primary perceptual division.

Norman nodded. The principle can be applied to any language-processing system, natural or artificial. Curious as it might sound, the Aymara-speaking software would, to a certain extent, think like an Aymaran. With its multi-valued logic, it would make distinctions that would never occur to a New York stockbroker; with its lack of grammatical gender, it would fail to make distinctions the stockbroker would unconsciously make. And Aymaran is only the easiest language, not the only one, to which the principle can be applied. Employing Guzman's translation techniques, it would be possible to develop front-end packages that, with the proper filtering out of Aymaran values and perceptions, would embody pure upper-class British perceptions or ancient Greek thought processes. We could examine the way judges in ancient Sumeria examined evidence.

That, Norman explained, was what they were up to at AFL: just as the developers of expert systems were trying to capture the knowledge of selected individuals in software, AFL was trying to capture the style of thinking,

the intellectual spirit of whole cultures. I mulled that over. Wouldn't there be a great advantage, I asked, in combining the two approaches, developing a system with a specifiable knowledge base and a specifiable style of thinking? Couldn't one develop, say, a machine with the knowledge of a high-energy physicist and the spirit of a 12th-century Mandarin? Or the knowledge of a modern statesman and the intellectual style of the first Continental Congress? But Norman suddenly looked uncomfortable and said that he couldn't discuss details of ongoing projects.

Reference Map

Golemics, Inc.
2600 10th St.
Berkeley, CA 94710
(415) 486-8347
Reader Service No. 101.

Micro Craft Corp.
4747 Irving Blvd., Ste 241
Dallas, TX 75247
(214) 630-2562
Reader Service No. 103.

New Functions
P.O. Box BB
Staten Island, NY 10302
(718) 273-2668
Reader Service No. 105.

Atari
1196 Borregas
Sunnyvale, CA 94086
(408) 745-2000
Reader Service No. 107.

Dr. Alan Winfield
Metaforth Computer Systems Ltd.
Unit 2 B The Newlands Center
Inglemire Lane
Hull HU6 7TQ
England
Reader Service No. 109.

Novix
10590 North Tantau Ave.
Cupertino, CA 95014
(408) 996-9363
Reader Service No. 111.

DFE Electronic Data Systems
USA: 5820 Stoneridge Mall Rd.
Suite 115
Pleasanton, CA 94566
(415) 847-2024
Europe: Singerstrasse 3,
7513 Stutensee
West Germany
07244-9011
Reader Service No. 113.

Real Time Devices, Inc.
1930 Park Forest Ave.
P.O. Box 906
State College, PA 16804
(814) 234-8087
Reader Service No. 115.

Interstate Voice Products
1849 W. Sequoia Ave.
Orange, CA 92668
(714) 937-9010
Reader Service No. 117.

STB Systems, Inc.
601 N. Glenville, Ste 125
Richardson, TX 75081
(214) 234-8750
Reader Service No. 119.

Univation
1037 North Fair Oaks Ave.
Sunnyvale, CA 94089
(408) 745-0180
Reader Service No. 121.

ANEX Technology, Inc.
151 North Route 9W
Congers, NY 10920
(914) 268-2400
Reader Service No. 123.

Edsun Laboratories, Inc.
7 Sears Rd.
Wayland, MA 01778
(617) 358-5667
Reader Service No. 125.

Digital Research Inc.
Box DRI
Monterey, CA 93942
(800) 443-4200
Reader Service No. 127.

Digitrol Computers Inc.
440 Phillip St.
Waterloo, Ontario
Canada N2L 5R9
(519) 884-4541
Reader Service No. 129.

RTCS Corp.

1390 Flynn Rd., Unit E
Camarillo, CA 93010
(805) 987-9781
Reader Service No. 131.

SoftLogic Solutions
530 Chestnut St.
Manchester, NH 03101
(800) 272-9900
Reader Service No. 133.

Phoenix Computer Products Corp.
1416 Providence Highway, Ste 220
Norwood, MA 02062
(617) 762-5030
Reader Service No. 135.

Morgan Computing Co., Inc.
P.O. Box 112730
Dallas, TX 75011
(214) 245-4763
Reader Service No. 137.

Genesis Microsystems Corp.
196 Castro St.
Mountain View, CA 94041
(415) 964-9001
Reader Service No. 139.

Level Five Research
4980 South A-1-A
Melbourne Beach, FL 32951
(305) 729-9046
Reader Service No. 141.

Human Edge Software Corp.
2445 Faber Place
Palo Alto, CA 94303
(415) 493-1593
Reader Service No. 143.

Poor Person Software
3721 Starr King Circle
Palo Alto, CA 94306
(415) 493-3735
Reader Service No. 145.

Minnow Bear Computers
P.O. Box 2233 Sta. A
Champaign, IL 61820-8233
(217) 398-6883
Reader Service No. 147.

Soft Advances
P.O. Box 49473
Austin, TX 78765
(512) 478-4763
Reader Service No. 149.

Selfware
3545 Chain Bridge Rd., Ste 3
Fairfax, VA 22030
(703) 352-2977
Reader Service No. 151.

Micro-Software Developers, Inc.
214½ West Main St.
St. Charles, IL 60174
(312) 377-5151
Reader Service No. 153.

Catalytix Corp.
55 Wheeler St.
Cambridge, MA 02138
(617) 497-2160
Reader Service No. 155.

Lattice C User's Group
P.O. Box 3072
Glen Ellyn, IL 60138
(312) 858-7950
Reader Service No. 157.

Whitesmiths
97 Lowell Rd.
Concord, MA 01742
(617) 369-8499
Reader Service No. 159.

The Software Toolworks
15233 Ventura Blvd., Ste 1118
Sherman Oaks, CA 91403
(818) 986-4885
Reader Service No. 161.

Smith & Smith Associates
P.O. Box 160
Hunt Valley, MD 21030
(301) 666-8129
Reader Service No. 163.

Vance Info Systems
2818 Clay St.
San Francisco, CA 94115
(415) 922-6539
Reader Service No. 165.

Kriya
505 N. Lake Shore Dr. Suite 5510
Chicago, IL 60611
(312) 822-0624
Reader Service No. 167.

Dennis Brothers' MacTEP
Berkeley Mac Users Group
1442A Walnut St., Ste. 153
Berkeley, CA 94709
Reader Service No. 169.

Red Ryder
The FreeSoft Co.
10828 Lacklink
St. Louis, MO 63114
(314) 428-8057
Reader Service No. 171.

The IM Underground
715 Church St. #16
Ann Arbor, MI 48104
Reader Service No. 173.

MacTutor
P.O. Box 846
Placentia, CA 92670
(714) 993-9939
Reader Service No. 175.

The Mac Street Journal
P.O. Box 6686
Yorkville Station
New York, NY 10128
Reader Service No. 177.

MacBriefs
P.O. Box 2178
Huntington Beach, CA 92647
(714) 842-0518
Reader Service No. 179.

MicroSparc
45 Winthrop St.
Concord, MA 01742
(617) 371-1660
Reader Service No. 181.

InCider
CW Communications/Peterborough
Peterborough, NH 03458
Reader Service No. 183.

Pandora Software
P.O. Box 55
Clearfield, UT 84015
Reader Service No. 185.

IEEE
345 East 47th St.
New York, NY 10017
Reader Service No. 187.

Aymara Fab Lab
1974 Gardner St.
Sunnyvale, CA 94086
Reader Service No. 189.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 198.**

Circle **no. 29** on reader service card.

Circle **no. 47** on reader service card.

Circle **no. 64** on reader service card.

Circle **no. 61** on reader service card.

Circle **no. 118** on reader service card.

Circle **no. 104** on reader service card.

Circle **no. 88** on reader service card.

Circle **no. 17** on reader service card.

Circle **no. 128** on reader service card.

ADVERTISER INDEX

Advertising Sales Offices

East Coast
Walter Andrzejewski (617) 567-8361

Midwest/West Central
Michele Beaty (317) 875-0557

Northern California/Northwest
Lisa Boudreau (415) 424-0600

Southern California/Southwest
Beth Dudas (714) 643-9439

Classified Advertising
Tim Ortiz (415) 424-0600

Advertising Director
Stephen Friedman (415) 424-0600

Advertising Coordinators
Jay Horvath (415) 424-0600
Alice Abrams (415) 424-0600

MOST SIGNIFICANT PRODUCT OF THE YEAR - PC WEEK

They said it couldn't be done. Borland Did It. Turbo Pascal 3.0

The industry standard

With more than 250,000 users worldwide Turbo Pascal is the industry's de facto standard. Turbo Pascal is praised by more engineers, hobbyists, students and professional programmers than any other development environment in the history of microcomputing. And yet, Turbo Pascal is simple and fun to use!

	TURBO 3.0	TURBO 2.0	MS PASCAL
COMPILATION SPEED	8.1	16	206
EXECUTION SPEED	9 SEC	13 SEC	20 SEC
CODE SIZE	12 K	12 K	35 K
BUILT-IN INTERACTIVE EDITOR	YES	YES	NO
ONE STEP COMPILE (NO LINKING NECESSARY)	YES	YES	NO
COMPILER SIZE	39K	35K	300K+
TURTLE GRAPHICS	YES	NO	NO
BCD OPTION	YES	NO	YES
PRICE	$69.95	$54.95	$295.00

(*) **Benchmark** run on an IBM PC using MS Pascal version 3.2 and the DOS linker version 2.6. The 179 line program used is the "Gauss-Seidel" program out of Alan R. Miller's book: *Pascal programs for scientists and engineers* (Sybex, page 128) with a 3 dimensional non-singular matrix and a relaxation coefficient of 1.0.

The best just got better: Introducing Turbo Pascal 3.0

We just added a whole range of exciting new features to Turbo Pascal:

- First, the world's fastest Pascal compiler just got faster. Turbo Pascal 3.0 (16 bit version) compiles twice as fast as Turbo Pascal 2.0! No kidding.
- Then, we totally rewrote the file I/O system, and we also now support I/O redirection.
- For the IBM PC versions, we've even added "turtle graphics" and full tree directory support.
- For all 16 Bit versions, we now offer two additional options: 8087 math coprocessor support for intensive calculations and Binary Coded Decimals (BCD) for business applications.
- And much much more.

The Critics' Choice.

Jeff Duntemann, PC Magazine: *"Language deal of the century . . . Turbo Pascal: It introduces a new programming environment and runs like magic."*

Dave Garland, Popular Computing: *"Most Pascal compilers barely fit on a disk, but Turbo Pascal packs an editor, compiler, linker, and run-time library into just 39K bytes of random-access memory."*

Jerry Pournelle, BYTE: *"What I think the computer industry is headed for: well documented, standard, plenty of good features, and a reasonable price."*

Portability.

Turbo Pascal is available today for most computers running PC DOS, MS DOS, CP/M 80 or CP/M 86. A XENIX version of Turbo Pascal will soon be announced, and before the end of the year, Turbo Pascal will be running on most 68000 based microcomputers.

An Offer You Can't Refuse.

Until June 1st, 1985, you can get Turbo Pascal 3.0 for only $69.95. Turbo Pascal 3.0, equipped with either the BCD or 8087 options, is available for an additional $39.95 or Turbo Pascal 3.0 with both options for only $124.95. As a matter of fact, if you own a 16-Bit computer and are serious about programming, you might as well get both options right away and save almost $25.

Update policy.

As always, our first commitment is to our customers. You built Borland and we will always honor your support.

So, to make your upgrade to the exciting new version of Turbo Pascal 3.0 easy, we will accept your original Turbo Pascal *disk* (in a bend-proof container) for a trade-in credit of $39.95 and your Turbo87 original *disk* for $59.95. This trade-in credit may only be applied toward the purchase of Turbo Pascal 3.0 and its additional BCD and 8087 options (trade-in offer is only valid directly through Borland and until June 1st. 1985).

NOT COPY-PROTECTED

TURBO PASCAL

Available at better dealers nationwide. Call (800) 556-2283 for the dealer nearest you. To order by Credit Card call (800) 255-8008, CA (800) 742-1133

Carefully Describe your Computer System!

Mine is: ___ 8 bit ___ 16 bit
I Use: ___ PC-DOS ___ MS-DOS ___ CP/M 80 ___ CP/M 86

My computers' name/model is: ______

The disk size I use is: ☐ 3½" ☐ 5¼" ☐ 8"

For update: original Turbo disk must accompany order

YES! I want the Best! Please send: Quantity

Pascal 3.0 $ 69.95 ___
Pascal w/8087 $109.90 ___
Pascal w/BCD $109.90 ___
Pascal w/8087 & BCD $124.95 (SAVE $24.90) ___

• These prices include shipping to all U.S. cities. All foreign orders add $10 per product ordered.

Subtotal (CA 6% tax) ___
Trade-in Credit Claimed: ___
Amount Enclosed: ___

Payment: VISA MC BankDraft Check

Credit Card Expir. Date: ___/___

Card #: ______

Name: ______
Shipping Address: ______
City: ______ Zip: ______
State: ______
Telephone: ______

COD's and Purchase Orders WILL NOT be accepted by Borland. California residents: add 6% sales tax. Outside USA: add $10 and make payment by bank draft, payable in US dollars drawn on a US bank.

11

BORLAND INTERNATIONAL

Software's Newest Direction
4585 Scotts Valley Drive
Scotts Valley, CA 95066
TELEX 172373

Circle **no. 14** on reader service card.